Reactive Robustness and Integrated Approaches for Railway Optimization Problems

Haahr, Jørgen Thorlund

Publication date:
2015

Document Version
Publisher's PDF, also known as Version of record

Ph.D. thesis

Reactive Robustness and Integrated Approaches for Railway Optimization Problems

Jørgen Thorlund Haahr

October 9, 2015

Dansk titel:
Reaktiv robusthed og integrerede løsningsmethoder til optimeringsproblemer i jernbanedrift
Type: Ph.d.-afhandling

Forfatter: Jørgen Thorlund Haahr

ISBN-nr:

DTU Management
Department of Management Engineering
Technical University of Denmark

Produktionstorvet, Building 426,
DK-2800 Kgs. Lyngby, Denmark
Phone: +45 45 25 48 00, Fax: +45 45 93 34 35
phd@man.dtu.dk
Oct, 2015

Summary

Planning railway operations is not a simple task as it entails solving multiple interdependent optimization problems. These problems have been subject to study in the literature for the last few decades, and are still profoundly researched. The robustness of a plan or schedule denotes the ability to absorb or withstand unexpected events such as delays. Making robust plans is central in order to maintain a safe and timely railway operation. This thesis focuses on reactive robustness, i.e., the ability to react once a plan is rendered infeasible in operation due to disruptions. In such time-critical situations, new plans must be found quickly. Integration of the different planning problems is also considered in this thesis as these problems are strongly interdependent in many cases. In contrast, finding feasible plans for each problem in isolation can lead to an overall infeasibility, e.g., during a disruption the updated timetable may be impossible to realize due to the lack of rolling stock units at certain positions. It is important to avoid creating problems for later or subsequent planning stages.

Several railway problems are studied in this thesis. The main contributions are summarized in individual chapters, some of which are papers that have been submitted to international scientific journals in operations research. The problems have been formulated as optimization problems and solution methods have been proposed to solve them using optimization theory and various solution techniques. In collaboration with industry and academic partners real-life and realistic data has been used to benchmark and test the solution methods.

A central actor and theme of the thesis is the rolling stock running on the railway networks. A public timetable is given, and in order to service the departures and passengers a rolling stock schedule (or circulation) is sought that provides the best compromise between operational cost, robustness, contract requirements and passenger satisfaction. In between train services the rolling stock units must be parked in the available depots. As trains cannot overtake each other easily, special attention must be given to avoid conflicting movements. Furthermore, rolling stock units are heavy and consume a considerable amount of energy in operation; with proper optimization tools a significant amount of the energy can be saved. A prompt optimization of individual train journeys helps the driver to drive efficiently and enhances robustness in a realistic (dynamic) environment.

Four international scientific prizes have been awarded for distinct parts of the research during the course of this PhD project. The first prize was awarded for work during the "2014 RAS Problem Solving Competition", where a freight yard optimization problem was considered. The second junior (PhD) prize was awared for the work performed in the "ROADEF/EURO Challenge 2014: Trains don't vanish!", where the planning of rolling stock movements at a large station was considered. An honorable mention (and second place) was awarded in recognition for *excellent work* in the "Discrete Optimization Challenge", where the aim was to minimize energy consumption in timetables. Finally, a second place was awarded in the "2015 RAS Student Paper Award", where a comparison of solution methods for planning shunting yard movements was considered.

Resumé (Danish Summary)

Planlægning af jernbanedrift er ikke en simpel opgave, da det indebærer at finde løsninger på flere indbyrdes afhængige optimeringsproblemer. Disse problemstillinger har der været forsket i de seneste årtier og der forskes fortsat aktivt. Robustheden af en driftsplan definerer i hvilken udstrækning planen kan absorbere eller være modstandsdygtig overfor uforudsete begivenheder fx forsinkelser. At lave robuste planer er centralt for at opretholde en sikker, pålidelig og punktlig drift. Denne afhandling har for det første fokus på reaktiv robusthed, d.v.s. evnen til at reagere når en plan bliver ugyldig fx ved driftsforstyrrelser. I tidskritiske situationer må en ny plan findes hurtigst muligt. Integrering af forskellige planlægningsproblemer er det andet fokus i afhandlingen, da de betragtede problemstillinger i flere tilfælde er indbyrdes stærkt afhængige af hinanden. At løse disse problemstillinger hver for sig kan resultere i en overordnet ugyldig plan, fx kan en nyfunden køreplan under en driftsforstyrrelse være umulig at realisere p.g.a. manglende rullende materiel på givne positioner. Det er således vigtig at undgå at skabe problemer for de planlægningsproblemer, som efterfølgende skal løses.

Flere problemstillinger indenfor jernbaneindustrien bliver undersøgt i denne afhandling. I de følgende kapitler opsummeres hovedbidragene, hvoraf flere udgør artikler, som er sendt til optagelse i international tidsskrifter indenfor operationsanalyse. Problemstillingerne er formulerede som optimeringsproblemer og løsningsmetoder er forslået ved brug af teori og teknikker indenfor matematiske optimering. Autentiske samt realistiske datasæt er i samarbejde med erhvervs- og universitetspartnere benyttet til at teste og sammenligne løsningsmetoder.

En central aktør og et centralt emne i denne afhandling er det rullende materiel, som kører på jernbaneskinnerne. En køreplan er offentliggjort, men for at servicere alle afgangene og passagererne søges en materielplan med det bedst mulige kompromis på tværs af driftsomkostninger, robusthed, kontraktmæssige krav og passagerernes tilfredshed. Imellem togtjenesterne skal det rullende materiel parkeres i de tilgængelige depoter. Tog kan ikke overhale hinanden i samme spor og en depotplan må derfor laves således, at togene kan komme ud uden være blokkeret af andre tog. Et rullende materiel er tungt og forbruger en væsentlig mængde af energi under kørselen; her kan et optimeringsværktøj kan medføre betydelige besparelser. En hurtig udregning til vejledning af individuelle togture kan hjælpe en lokomotivfører til at køre energibesparende og forbedre robustheden i et realistisk (dynamisk) miljø.

Fire internationale priser er modtaget undervejs, i dette PhD projekt, for forskellige dele af forskningsprojekterne. Første plads blev tildelt under "2014 RAS Problem Solving Competition", hvor rangering og sortering af fragttog blev undersøgt. En anden plads blandt PhD holdene blev tildelt under "ROADEF/EURO Challenge 2014: Trains don't vanish!", hvor planlægningen af det rullende materiel på en større station blev taget i betragtning. En hæderlig omtale (honorable mention) og anden plads blev tildelt under "Discrete Optimization Challenge" for fremragende arbejde (excellent work). Her var målet at minimere energiforbruget ved planlægning af køreplaner. Endeligt blev en anden plads tildelt under "2015 RAS Student Paper Award", hvor forskellige løsningsmetoder på depotplanlægningen blev undersøgt.

Preface

This thesis has been submitted to the Department of Management Engineering, Technical University of Denmark (DTU) in partial fulfillment of the requirements for acquiring the Doctoral of Philosophy (PhD) degree.

The PhD project is part of the RobustRailS project, which is a large interdisciplinary project at DTU aiming at improving railway operations. The project has been co-founded by the Danish Council for Strategic Research. The study has been conducted, in the period between August 2012 to October 2015, mainly at DTU Management Engineering but also in close collaboration with the Danish State Railways (DSB) and Cubris. The DSB is the largest train operating company in Denmark and only transports passengers on railway infrastructure. The DSB provides intercity train services but also operates the suburban railway network, S-train (S-tog), in the greater Copenhagen area. Cubris is a technology company that focuses on solutions for the railway business, including development and deployment of train speed advisory systems to railway operators. The PhD project has been supervised by Professor David Pisinger and Professor Jesper Larsen.

The thesis deals with several topics of railway operations in operations research. The first part gives an introduction and overview to the area of research. The second part lists research papers, submitted to international per-reviewed journals, and technical reports. All papers have been co-authored. A concluding chapter summarizes the thesis.

Kongens Lyngby, Denmark, October 2015

Jørgen Thorlund Haahr

Acknowledgments

This thesis has consumed a lot of time and effort during the past 3-4 years, and would not have been what it is today without the support and invaluable contribution of many talented, enthusiastic, capable and caring people.

First, I want thank my main supervisor Professor David Pisinger for trusting and giving me the opportunity to enroll as a PhD student under his supervision. For your lasting optimism and usable feedback throughout this period. Thank you for the help lifting the quality of this PhD through your guidance and advice. Many interesting opportunities have appeared in form of international competitions and external collaboration, I thank you for opening doors and supporting me in these situations. Next, I also want to thank my co-supervisor Professor Jesper Larsen for helping on a variety of things including guidance and proofreading.

My colleagues at DTU Management Engineering have my gratitude for engaging in interesting discussions and in general for contributing to a great working atmosphere. A special thank you to the people who have become my co-authors through research projects. In particular, I must thank Associate Professor Richard Lusby who co-authors many of the papers. For your untiring work, feedback, suggestions and proofreading. Together we managed to win first prize of the INFORM RAS Problem Solving Competition 2014. Despite all the hard work we have had a good time.

I also want to thank Professor Leo Kroon for hosting me at the Rotterdam School of Management at Erasmus University. Thank you for providing a great working environment, valuable feedback and engaging in joint work together with Joris Wagenaar and Lucas Veelenturf.

Without the financial support of the Danish Council for Strategic Research, who funded the RobustRailS project, my PhD study would not have been possible. I thank them for the support and also for making the RobustRailS group possible. Through many scientific and partner meetings this interdisciplinary forum has widened my understanding of railway operations and provided feedback to my research.

Thanks to the Danish State Railways (DSB) and my main contact, Julie Jespersen Groth. Without them, I would not have been able to test and verify my research using real-life data. A great deal of railway specific knowledge would not have been available to me without their support. In addition, I want to thank Curbis ApS for letting me work on an interesting research project optimizing train energy efficiency. It has been a pleasure and fruitful to work in close collaboration with industrial partners.

At the bottom on of this list, but at the top of my heart, I am truly grateful for the love and support of my wonderful wife, Amalie, and daughters, Johanne and Regine. Thank you for your patience, support and encouragement during this PhD study. For allowing me to work long hours when needed, and at the same time reminding me of the important things in life.

Contents

Part I

Introduction

Chapter 1

Introduction

Railway operation is a complex planning problem that requires tight synchronization on a minute level. Operating an efficient timetable requires careful and detailed planning of existing resources such as rolling stock, crew and infrastructure. The complexity of the planning has lead to a problem decomposition into different layers. Depending on whether the aim is to serve long-term or short-term needs, different levels of abstractions are used to make the problem more manageable. A separation of concerns is also adopted where interdependent planning-aspects are solved using a top-down or sequential approach, thus possibly introducing infeasibility or suboptimal plans in practice.

Trains, pulled by a mechanical engine, have been around for more than one century, and the idea of using rail-tracks dates even further back. Today, large rail networks connect cities of different countries. Large investments continue to improve the railway transport industry even though cars, vessels and aircrafts serve as alternative modes for passenger or freight transportation. A comparison between different modes shows that railway passenger transportation is still an energy-efficient mode of transportation [11], and that rail freight transportation is only surpassed by water-borne modes. Continuing to invest, optimize and make this large industry as attractive as possible is therefore justified. Both goals can be achieved using an improved or a more comprehensive planning process that minimizes operation cost and optimizes passenger convenience. A number of such railway planning problems are the subject of this thesis.

The comprehensive railway planning problems are usually time-consuming and difficult to solve. Some problems may contain too many feasible options, making it difficult to find a high quality plan, while others may be too constrained to find anything feasible. A constant need of better and faster methods is therefore highly motivated. Computers are able to execute more than one billion (10^9) instructions per second and can be useful for helping explore the solution space. Compared to human planners, different search strategies can be explored extremely fast. Thus, allowing exploration of many more solutions and allowing more comprehensive methods. However, in many cases, an exact mathematical model cannot describe the problem at hand completely, therefore motivating a combination of both computers and planners. The computer can function as a decision aid system that allows planners to evaluate plans more efficiently, e.g., in terms of time spent or in terms of solution quality.

Operations Research is a sub-field of mathematics that is concerned with problem solving and decision making with the use of mathematical modeling, statistical analysis and optimization techniques. With an original breakthrough in military applications during the second world war, this field has been applied to a variety of other (civil) problems such as transportation logistics, scheduling and infrastructure investment. Over a few decades, with the development of computers, Operations Research has been successfully applied to many different problems including railway operation. Despite the arrival of increasingly more efficient computers, some optimization problems

still remain too difficult to solve completely, i.e., to optimality. Different solution methods must be explored in order to find an eligible approach to each individual problem. Ideally, an optimal solution is sought, and many such *exact* solution methods have been suggested to solve various problems. However, in practice, planners are not willing to wait indefinitely long to find the best solution. If the problem cannot be solved within reasonable time a *heuristic* approach may be more eligible. In contrast to *exact* solution methods, *heuristic* approaches aim at finding high quality solutions using less computational time at the expense of proving optimality. A number of railway optimization problems will be considered in this thesis, where an important goal is to find efficient solution methods. If not possible to find the optimal solution within reasonable time, then the goal is to find the highest solution quality possible within the given time limits.

The difference between a feasible solution, e.g. found manually by a planner, and a solution found using an optimizer can be substantial. Depending on the objective, this can for example be directly translated to higher savings, better utilization of resources or more robust plans. The value of proper optimization planning tools can easily be crucial to operations in a competitive market. Planning problems are handled offline where time is a luxury, and every detail can be resolved. In practice, on the day of operation, things usually never go *exactly* according to the plan. Minor disturbances are natural elements of operation and can become a nuisance when things start to escalate. Rescheduling, in order to get everything back on track, is another variant of the planning problem. However, the solution times of the optimization methods are now paramount, and a good solution is required to cut losses and minimize overall inconvenience. With limited time available, heuristic approaches may be the only viable choice. A number of the considered problems in this thesis are rescheduling problems that need to be solved continuously throughout the planning horizon.

Optimal or high quality plans, e.g. found by an optimizer, are prone to being non-robust. The available, and in particular the critical, resources are nearly fully utilized in high quality plans, consequently leaving no slack, i.e., the ability to make small changes if needed. A degree of uncertainty exists in most planning problems and ignoring the uncertainty can result in undesired (and potentially cascading) negative consequences. In railway operation, delays and disruptions are unfortunately familiar concepts, thus making plans that are robust a necessity. *Reactive* robustness is a central theme of this thesis, i.e. the methods investigated are in general fast enough to find recovery plans. Only sufficiently fast method are eligible as decision support must be provided before it is too late to act. *Proactive* robustness, i.e. planning reasonable slack or resilience in the resource utilization, is also considered to some extent.

1.1 Thesis Overview

A number of different railway optimization problems are studied in this thesis and covered in individual chapters. The thesis consists of four main parts. In the first part, Part I, the considered railway problems are introduced and put in perspective before specific problems are considered. The thesis is concluded in the last chapter (Chapter 3) and possible future research directions are given. The second part, Part II, investigates Rolling Stock and Depot Planning problems. In Part III a large-scale freight-yard problem is considered. The forth part, Part IV, is dedicated to energy optimization in railway. In particular the generation of train speed profiles and synchronization in a timetable.

Part I

An overview introduction to railway optimization is given in Chapter 2 of Part I. The main focus is on introducing the problems considered in the papers. Other relevant or related problems are mentioned or described briefly in order to provide an overview of the bigger picture.

The railway problems are motivated and introduced informally using examples to present individual optimization problems. Problem-specific robustness concepts are discussed for the individual problems.

Part II

Five different research problems concerned with planning of rolling stock are considered in Part II. The first two chapters consider variants of assigning rolling stock to train services for a day of operation. The next chapter considers the problem of parking rolling stock units between arrivals and departures. The following chapter considers integration of rolling stock planning and the parking on depot tracks. The last chapter takes on a station-view of the rolling stock problem where the problem is to match arrivals and departures at a train station.

In Chapter 4 the Rolling Stock Scheduling Problem is investigated. A solution approach based on column generation embedded in a Branch-And-Price (BAP) framework is presented. A complete branching strategy is presented, and proven to be complete. Compared to previous research in this area this method assigns one rolling stock route to every available train unit. The considered model brings the actual day-to-day problem closer by allowing unit-specific constraints. Both planning and disruption scenarios, provided by the Danish State Rails (DSB), are benchmarked and the method is able to find high quality solutions within a short time-frame. An additional benchmark demonstrates that maintenance constraints can be incorporated without significant run-time penalties.

An extension of the Rolling Stock Scheduling Problem is considered in Chapter 5. In addition to finding rolling stock routes, there are now also restrictions on how train compositions may change during the day. Depending on business rules and physical limitations, a set of feasible transitions is explicitly disallowed. An extension of the model in Chapter 4 is presented and solved in a BAP framework. This framework is compared to a state-of-art MIP model that only considers anonymous flow. Benchmarks, including planning and disruption instances, provided by the DSB and the Netherlands Railways (NS), show that the model extension can be solved to optimality within short time. The state-of-art MIP model, solved using CPLEX, is solved faster but outputs only anonymous rolling stock flow.

Parking rolling stock units in a conflict-free way can be difficult when depot track resources are scarce. In Chapter 6 we revise the Depot Planning and Matching Problem. The problem entails matching arriving and departing train units and assigning them to appropriate shunting tracks in a conflict free manner. We present a comparison benchmark including multiple, some novel, solution approaches: a Mixed Integer Linear Program, a Column Generation framework, a Constraint Program, a randomized greedy heuristic and a two-phase decomposition method. The benchmark contains multiple real-life instances provided by the Danish State Rails (DSB) and Netherlands Railways (NS). The second place of the "2015 RAS Student Paper Award" was awarded for this work.

Integration of rolling stock schedules and depot parking is considered in Chapter 7. Solving these problems in isolation is unattractive if depot track resources are limited. An iterative framework for solving this problem is presented. Realistic problem instances are solved within a few minutes, making the approach eligible for short-term planning. The depot planning problem is a submodule that is given some consideration and discussion. Interestingly, one of the results shows that previous research in the literature using column generation is to some degree incorrect.

The Train Departure Matching Problem is investigated in Chapter 8. The problem is an extension of matching problem (considered in Chapters 6 and 7) that entails matching arriving and departing trains at a large train station. Rolling Stock units of compatible types are interchangeable, however, sufficient distance-to-maintenance and time-to-maintenance must be available prior to departure in order to perform the specified round-trip. The presented problem is a sub-problem of

the ROADEF 2014 challenge (*Trains don't vanish!!!*) [2]. This is an extended matching problem that we prove to be $\mathcal{NP}$-hard in the strong sense. Two exact solution methods are presented. First, a MIP-based approach that is solved using CPLEX. Second, a column generation approach in a heuristic setup. The second junior (PhD teams) prize was won using the former solution method as a solver in a larger framework. A technical report of the complete solution method for the challenge is presented in appendix A.

Part III

In chapter 9 a matheuristic is presented for solving the Railroad Hump Yard Block-to-Track Assignment problem. Hump yards are consolidation points for arriving and departing freight trains, where an efficient processing of railcars is paramount. The problem is introduced in the RAS (INFORMS) Problem Solving Competition 2014 [1]. In Chapter 9 the problem is decomposed into three sub-problems that are solved, to some degree, in isolation. The solution framework processes the large data instances using a time-sweep approach. A lower bound is found in order to evaluate the heuristic solution method. The benchmark shows that the solution method is able to find good solutions within several minutes. Additional *what-if* studies are investigated and show that a significant improvement in terms of solution quality can be achieved if trains can be delayed up to a few hours. The solution method is also sufficiently fast to be used as a rescheduling tool on a daily basis, e.g., when the demands or inbound train compositions change. The reasonably fast response times makes the solution method a reactive robust method. The first prize award for the competition was won using the presented solution method.

Part IV

The Train Speed Profile problem is considered in Chapter 10. Given a train journey between two full stops the goal is to minimize the total energy consumption subject to various constraints. Train are very heavy and consume a considerable amount of fuel or electricity when accelerating. Even small percentage savings per journey can lead to significant financial saving or a smaller environmental impact. A novel Dynamic Programming solution method is presented and solved using a label setting algorithm. It is argued that the method is flexible and does not, in contrast to other methods in literature, rely on advanced analytical mathematics. In addition to gravitational pull and multiple speed limits, the method also models passage points. No existing work in literature has been identified that models passage points. A benchmark using real-life instances shows that the method is efficient and sufficiently fast to be used for real-time driving assistance. In combination with an on-board Driver Advisory System (DAS) this method can improve the overall robustness by helping the train driver to arrive on time at the destination, even in the face of sudden updates to the journey. In addition, the passage point extension can be used to facilitate a more robust train journey by forcing the train to arrive timely at checkpoints, thus reducing the risk of falling behind or ahead of schedule.

Minimizing individual train speed profiles does not necessarily reduce operational costs. In some practical billing contracts [16] the goal includes minimizing energy peaks. In Chapter 11 the problem of synchronizing train departures is considered. The considered problem is introduced in the Discrete Optimization Challenge organized by the Friedrich-Alexander-University Erlangen-Nürnberg [3]. Under the assumption that the travel times between two stations are fixed (with an associated speed profile), the goal is to minimize the maximum energy consumption over uniformly distributed periods in the considered horizon by deciding the departure times for the train services. Trains can regenerate power through braking, but the total consumption for every second can never be negative since all excess power is considered lost. A MIP approach and several matheuristics are presented. The benchmarks show that high quality solutions are found within reasonable time.

An honorable mention (and second place) was awarded for the excellent and high quality work performed.

Chapter 2

Railway Optimization Problems

The railway industry faces many different planning problems that essentially are (or can be formulated as) optimization problems. Typically many so-called *planners* (employees at a railway company) put many manhours into making detailed schedules for different aspects of the railway operation. In this chapter, an overview of the traditional optimization problems is presented. A selected number of problems will be introduced and discussed in greater detail. A more comprehensive review of railway optimization problems is presented by Caprara et al. [9].

2.1 Planning Horizon

A coarse partitioning of railway optimization problems can be made based on the scope and goal. For example, the level of detail required (or that is practical to consider) depends on whether the aim is to determine fleet investments or whether to determine the next day's operation. The goal and restrictions of the planning depends on whether the outcome is to provide strategic, tactical or real-time decisions. To some degree the same underlying problems are solved but the main difference is the extent or level of detail and which basic assumptions are made. Multiple layers of abstractions are usually adopted, serving the different needs in different contexts:

Strategic decisions are long-term choices that can be changed later. Revising these choices can, e.g., incur significant financial loss or be inconvenient to several involved parties such as the passengers. Strategic decisions have large consequences for the next years of operation and choices must match the expected future demand among other operational constraints. However, the future is uncertain, making it impractical to plan to every detail. Strategic choices rather define the overall guidelines. Examples of strategic planning problems include:

- Planning a new timetable or negotiating a new service contract. Line structures are determined and baseline choices are made that must be followed in the future realization of the plan.
- Decisions on making rolling stock investments, which have a huge financial impact. New product lines of rolling stock must be purchased in order to replace existing units that are close to being obsolete.

Tactical decisions are concerned with the imminent future, say choices affecting a few months to a year in advance. The strategic guidelines are adopted but a number of things are still very uncertain. In the tactical phase a more detailed plan is required such that public timetables can be released. Examples of tactical problems include:

- Proactive maintenance is scheduled a few months in advance thus requiring modifications to the normal timetable. Changes to the planned timetable are necessary in order to allow the maintenance crew to work on track equipment during operation.
- It is impossible to predict the position of every rolling stock unit even a few days in advance, but rolling stock circulations are determined. These circulations can, e.g., ensure smooth unit movement and regular in-time maintenance appointments.
- A crew roster plan can be determined that outlines which driving tasks to perform in a single duty. Together all tasks are covered and individuals can be assigned to rosters.

Operational plans are made only a few days in advance setting all remaining details into place. The tactical plan is followed as closely as possible, but all irregularities must be handled. Examples of operational problems include:

- Assignment of physical rolling stock units to circulations. Unplanned activities such as washing graffiti or early maintenance have to be taken into account.
- Rescheduling of train drivers in order to cover unexpected illness and other unplanned events.

Additional phases may exist or be defined differently depending on the practice of the considered railway operator. For example, a significant amount of research has been conducted in disruptions management. Disruptions occur during the day and new plans must be made in real-time. Groth et al. [14] and Kroon et al. [19] give an overview of the disruption management process in railway.

2.2 Robustness

On November 14, 1957 Dwight D. Eisenhower repeated a statement he had heard in the Army: "Plans are worthless, but planning is everything". Plans are useless due to uncertainty as everything will not unfold exactly as planned. However, planning is essential as we need to be familiar with the planning problem.

The future is uncertain and it is consequently optimistic to expect a given plan to be tenable to every detail. Few or no planning problems are independent of future uncertainty, and planners must therefore consider how to address the uncertainty. In contrast to Eisenhower's statement, a plan is sought that to some degree is *robust* or *resilient* against uncertainty. Naturally, it is impractical if not impossible to plan for all possible future events, nevertheless, some stochastic events can be handled. The remedy most often comes at a cost as uncertainty is accommodated by measures such as imposing resource slacks, inserting buffer times or adding redundancy. The goal is to find a reasonable trade-off between economical loss and gained *robustness*.

In railway optimization literature the term "robust" is widely accepted and adopted when dealing with uncertainty. What constitutes the robustness of a plan can be debated, in addition to which modeling adjustments are to be made in order to enhance it. A robust plan is of paramount importance in railway optimization as it aims to avoid unwanted or costly consequences of common deviations such as delays. If things do not go according to plan the consequence is often significant: delays in a timetable can have a rippling effect, and a *left behind* rolling stock unit will not only disrupt local depot plans but possibly also inflict a capacity shortage somewhere else.

General notions of robust optimization exist, however in railway literature robustness is most often specific to the application. For the timetabling problem train headway, buffer or run-time surplus can be considered to enhance the overall robustness. Additional time gives trains an opportunity to catch up, if they are delayed. In crew scheduling a depot with standby drivers efficiently reduces the risk of canceling trains due to sickness, delays and other factors. Adding

buffer time to rolling stock shunting operations reduces the risk of delays in the face of unexpected events such as equipment failure. In other optimization contexts the objective is to minimize the expected cost given a certain set of failure scenarios, e.g., minimizing communication network capacity requirements in the face of single link failures [15].

The cases mentioned in the last paragraph are examples of *proactive* robustness, i.e., ensuring that the overall plan is guarded or persistent against uncertainty such as delays. The overall plan is unchanged as no change or re-planning is required. *Reactive* robustness is a different category of robustness that focuses on recoverability, i.e., how to recover from an unexpected event such as a disruption. For example, if some part of the railway infrastructure becomes temporary unavailable then the problem is to determine how to update the plans (e.g. timetable, rolling stock schedules and crew schedules) in order to avoid infeasibility causing unneeded cancellations. A sub-goal is to return to the original plan as quickly as possible, and to follow the original plan as closely as possible during the disruption. In contrast to proactive robustness, the remedy is a change of the overall plan (or a portion of it) and not built-in buffer times or redundancy. A recovery plan can be computed a priori or can be computed on demand. The former is faster as it is readily available during a disruption, but it is not always possible due to the vast number of possible things that could happen, e.g., consider the number of possible different locations, durations and starting points for an infrastructure failure. Precomputing all such plans is impractical since the actual positions of available rolling stock units or train drivers is unknown until a few days in advance. A recovery method (or framework) ensures swift and high quality recovery of unexpected events, thus increasing the overall robustness of the system.

In this thesis the focus is on reactive robustness. In general the considered solution methods or frameworks are responsive, i.e. eligible as online decision-aid systems to handle or re-schedule planned schedules when new information is available. The computational time requirements of a solution method is a key characteristic for any recovery strategy; if a short duration of time available to react, then even less time can be spent by a solution method.

2.3 Timetabling

The railway timetable is the cornerstone of all railway planning problems. The timetable determines the level of service and also imposes many restrictions or requirements on available resources such as infrastructure, rolling stock fleet and crew base.

Figure 2.1 shows an example of a timetable. The shown timetable specifies the operation of the **E** line between Holte and Køge during daytime, Monday through Friday. This diagram shows all scheduled (hourly) departure times for the passengers convenience. Figure 2.2 shows an overview of all lines in the S-train network in Copenhagen.

Passenger railway trains usually have priority over other traffic on the infrastructure, and it also constitutes the majority of the traffic in European countries. The infrastructure is, however, shared with freight trains, which may also drive according to their own timetable. Allocation of infrastructure is needed in advance, but compared to passenger trains, freight trains may have less strict requests regarding departure and arrival times. Some freight operators do not use timetables but rely on a more on-demand approach.

The timetable, from an infrastructure owner's perspective, is a complete list of all train traffic in the network. This total overview provides predictability, and is used for traffic control. Trains have allocated infrastructure paths that allows safe (or conflict-free) passage through the infrastructure. Details such as headway buffer times, track segment allocations, platform assignment and signaling are resolved in advance to allow seamless operation.

Infrastructure models are used in order to check whether a timetable is possible, and to measure whether it is a good utilization of the capacity. Both macroscopic and microscopic models

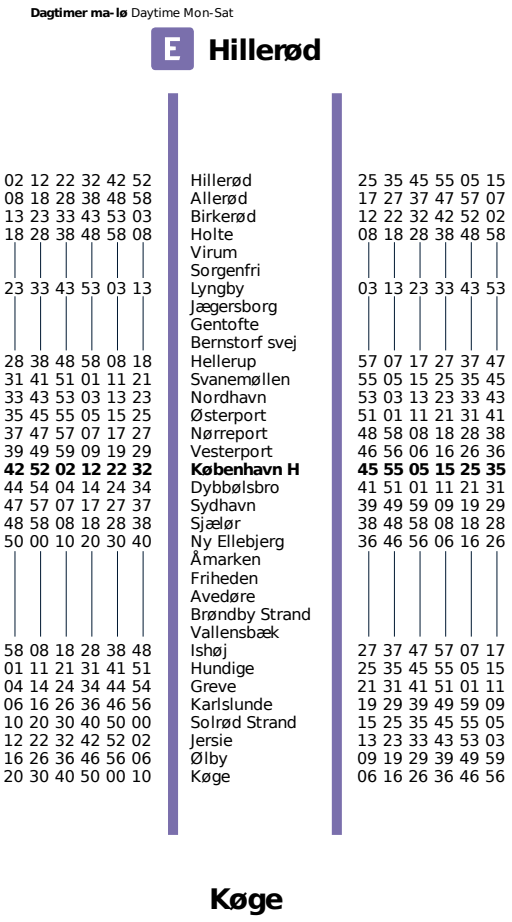


Figure 2.1: The diagram shows the public timetable of line **E** operating in the S-train network in Copenhagen during daytime, Monday through Saturday. The hourly departure times between the terminal stations (Hillerød and Køge) are shown.

have been studied in the literature. Due to the large number of elements such as track-segments, crossing and signals, a complete representation of infrastructure may be impractical to adopt for high-level decisions. Macroscopic modeling reduces the complexity by aggregating stations, lines or junctions. Using this level of detail it is possible to search for train paths and making sure that the fundamental requirements and resources are available. However, the actual routing through the aggregated nodes (e.g. a station) requires a microscopic model of the system, including interlockings, tracks, signals and more. An introduction and overview of timetabling and infrastructure modeling is given in [16].

Following a given timetable to the letter can prove difficult during operation. Trains can be delayed due to passenger irregularity, passenger crowdedness, signaling errors, telecommunication failure, weather conditions et cetera. The mentioned primary delays can lead to secondary knock-on delays. A delayed train can block other upcoming trains that uses the same part of the railway infrastructure. Whether or not mentioning robustness, much research [7] has studied the effect of allocating buffer time in order to make the timetable more robust. Small delays can be absorbed by making use of allocated buffer time, which otherwise is left unused, thereby avoiding knock-on effects.

Figure 2.2: A map of the S-train line network in Copenhagen. Each color represents individual lines. All intermediate stations are illustrated using white dots on the lines.

2.4 Rolling Stock Scheduling

Rolling stock units are needed to service the train services prescribed by the timetable. The Rolling Stock Scheduling Problem (RSSP) consists of assigning the available fleet to the services as best as possible given some operational constraints.

The type of rolling stock is decisive when determining models for solving the problem. First, some rolling stock systems are self-propelled while others may require locomotive engines to pull them. This inspires two branches of research. A reference model for the former is presented by Schrijver et al. [24], and a model for assigning locomotives to carriages is presented by Cordeau [10]. In order to fit the demand more closely, rolling stock is combined and split during operations in some networks, whereas in other systems exactly one unit can be assigned to every trip service, e.g., a metro. The former variation is more generic and difficult to solve as the assignment must also consider tradeoffs between passenger demand and operational expense, and likely also provide additional rules for changing the compositions. Alferi et al. [5] provide a base model for this problem. The rolling stock planning problems considered in this thesis assume self-propelled units that can be split and combined (i.e. decoupled and coupled) in operation. The problem structure is simplified if exactly one unit can be assigned to each trip service. A fleet of compatible self-propelled units is

beneficial in contrast to heterogeneous, incompatible or traction-less units. A homogeneous fleet results in an increased robustness as they provide great flexibility. With cockpits on both sides of a train, the orientation of the train is not limiting, and with rolling stock compatibility, units are interchangeable and thus able to absorb minor short-term or real-time deviations in the rolling stock plan.

A simple variant of the RSSP has similarities to the Multicommodity Flow Problem (MCFP) [4]. A set of trips $\mathcal{T}$ is given that needs to be serviced, where very trip departs from station, $a \in \mathcal{S}$, at time t_a and arrives at station, $b \in \mathcal{S}$, at time t_b. A fleet of rolling stock units are available that can service these trips. The core problem is to find the minimum cost schedule. Costs are usually related to the demand shortage, total mileage (energy consumption, infrastructure cost and maintenance tear-and-wear), end-of-day balance deviations and the number canceled trips. A time-space network can be built that represents the set of trips, see Figure 2.3. Note, that in some cases trips have a natural predecessor and successor trip, thus defining trip sequences. By construction, when following a trip sequence a rolling stock unit never needs to be coupled or decoupled, i.e., moved in or out of a shunting area. However, whenever a unit enters or exits a trip sequence, it will require the unit to be coupled onto or decoupled off an existing train composition. Coupling (or decoupling) trains during the day poses an operational risk. Although the operation is supposed to be seamless, unexpected errors can occur. In order to accommodate this issue (thereby improving robustness) a minimum separation time can be set between any two coupling operations, thus allowing some buffer time to handle some potential problems. In Figure 2.3 the example solution shows the assignment of three different rolling stock units. The largest unit needs to perform three shunting movements. It is coupled at **København** when entering an existing composition, and decoupled later at **København** when exiting. Afterwards it performs one trip on a second trip-sequence where it again needs to be coupled at **København**.

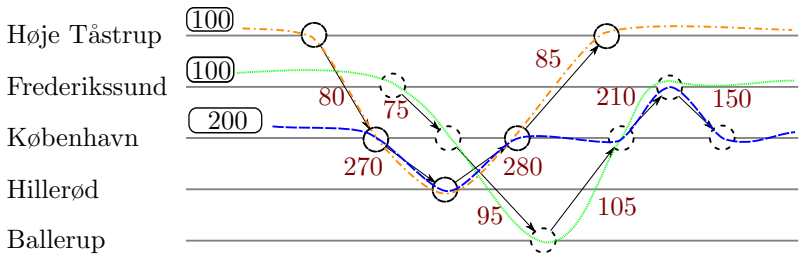


Figure 2.3: Example of a small rolling stock problem, and an example solution. Stations are shown on the vertical axis, and time is shown in the horizontal axis. Each trip has been assigned an integer demand. The initial positions of the available rolling stock is shown next to the stations.

Similar to the MCFP, a network of arcs (trips) and a fleet of non-splittable commodities (rolling stock units) are defined. The goal is now to find a minimum cost flow, where the arc-costs can be both negative (demand cover) and positive (operational expense). Bounds may be imposed on arcs in order to limit the resulting composition size because it cannot exceed the available platform length at the stations. From here on, the similarities with MCFP diminish:

- The total number of couplings and decouplings must be considered
- End-of-day balance deviations must be considered
- The positive demand arc cost becomes zero once all demand is covered.
- Uncovered arcs incur a cancellation penalty.

Unit Specific Constraints

Rolling stock units are classified by the rolling stock types, e.g., the Copenhagen Suburban Railway Operator (DSB S-tog) operates two types, **SA** and **SE**. The units are self-propelled and similar in most aspects; the governing difference is the passenger capacity, and naturally also weight, length and operational cost. The train units are compatible in the sense that both types can be combined arbitrarily to form larger train compositions.

When modeling rolling stock units are typically aggregated by their type. The aggregation makes modeling simpler and possibly also the solution method more efficient. From a strategic or tactical point of view, this assumption is easy to justify as the exact positions of units are impossible to know. An example of such modeling is presented by Fioole et al. [12].

In operational planning more details are known, such as the fleet position and maintenance status. The assumption that all units of the same type are interchangeable no longer holds as some units, e.g., have few miles or a small amount of time left before they must be maintained.

Rolling stock maintenance constraints have been considered by several authors in literature [10, 25], and the relevance is therefore apparent. At DSB S-tog different types of maintenance must be performed, some of them relate to the driven mileage while others to time. Firstly, regular maintenance is required just before e.g. 50, 150, 300, 800 and 1800 mega-meters. Secondly, a sand inspection, used for the wheel slip protection system, is required after every 10 mega-meters. Thirdly, during winter anti-icing must be applied every seventh day.

Other use cases for unit specific constrains are listed below:

Commercial Coating: Rolling stock units can be coated with commercials that generate additional revenue to the railway operator. At DSB S-tog the customer requires such trains to run on specific lines or on specific trip segments.

Equipment Failure: Due to equipment failure a self-propelled unit may have a reduced acceleration or braking capacity. Such units can still be used in service but only when coupled with other units.

Unit specific constraints are easily incorporated if units routes are modeled explicitly, in contrast to modeling anonymous unit types. In chapter 4 a Branch-and-Price (B&P) framework is presented that explicitly assign routes to rolling stock units. Maintenance constraints are included in order to demonstrate the framework.

Composition Transitions

An important extension to the presented RSSP is limiting the allowed composition transitions. Each trip in the timetable is assigned one composition, which is defined as the *ordered* set of rolling stock units. Figure 2.4 shows examples of different compositions using two different types of compatible rolling stock.

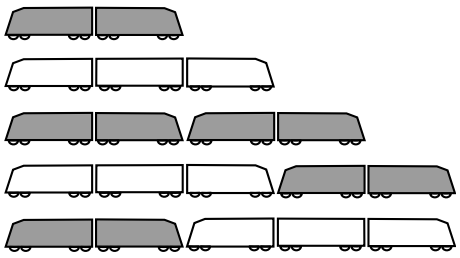


Figure 2.4: Examples of different possible train compositions with two different (compatible) rolling stock types.

Depending on the physical layout of tracks and business rules, restrictions are imposed on how compositions can be changed (i.e., upgraded or downgraded) between two trips in the same trip-sequence. A few simple examples are listed:

- At a station without any adjacent depot tracks, the composition of the first and following trip must remain identical. No facilities exists to perform a composition change.
- At stations where the train must exit in the same direction as it entered, the composition of the following trip is the reversed.
- At stations with adjacent depot tracks to the front of the train, units can only be added or removed from the front end.

More formally, a transition graph can be constructed that defines the feasible choices. Figure 2.5 illustrates a transition graph for one trip sequence. The graph is essentially a state diagram representing all feasible transitions between states. A path from the source to the sink represents a feasible chain of composition. The transition rules reflect what is doable or considered rational during operation, e.g., uncoupling the middle unit in a compositions with three units is not preferred due to the complexity of performing such a task. Therefore, the transition rules also ensure a more smooth and therefore a more robust plan.

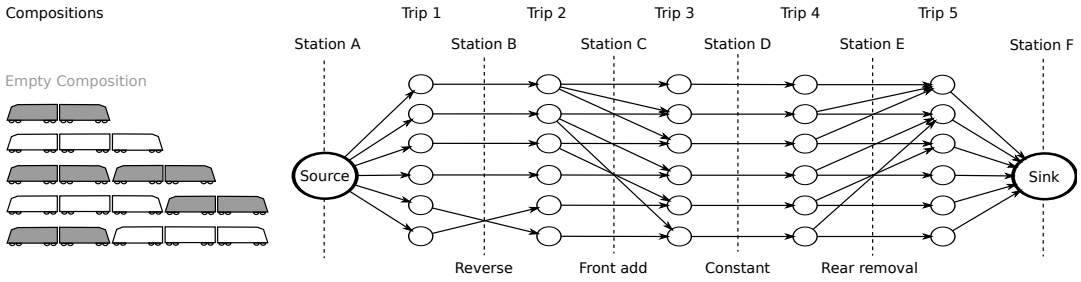


Figure 2.5: An example of a transition graph for a trip sequence. The arcs represent valid transitions between two consecutive trips.

Alfieri et al. [5] present a variant of the RSSP with composition transitions. The presented MIP model has good integral properties, and has been successfully adopted for further extensions in the literature [8, 12, 21].

One drawback of the MIP model is the anonymous modeling of rolling stock units. Feasible composition transitions are found and a feasible fleet flow is determined; however, the individual rolling stock routes are not explicitly handled. Consequently, imposing unit-specific constraints such as maintenance is not directly possible. Indirectly, maintenance constraints can be modeled by adding these as their own unit-types [25]. This approach is however not scalable. In Chapter 5 a column generation approach is presented that models unit routes explicitly while considering composition transitions. Ultimately, rolling stock units are not fully interchangeable. A more detailed plan for the rolling stock units implies less uncertainty and arguably therefore also a more robust plan.

2.5 Depot Matching and Parking

A rolling stock schedule implicitly determines when units are going in or out of the shunting yard. The shunting yard is here considered to be a depot for storing trains; when a train is not in service it has to stay (or wait) on some depot track. A depot track can be accessed from one or two directions depending on the infrastructure, and a train can naturally not overtake another train on the same track. Depot capacity is in some cases a scarce resource that requires careful

utilization. An otherwise feasible rolling stock schedule may be impossible to execute due to depot capacities.

The Depot Parking Problem (DPP) is the problem of finding a feasible assignment of arriving (and departing) rolling stock units to depot-tracks, subject to a number of constraints. In the simple form it is assumed that units arrive (or depart) one at the time, in isolation. Thus the constraints solely consist of ensuring that each individual depot track length is not violated, and ensuring that the train units are not blocking each other on the depot tracks. Figure 2.6 illustrates the core of the problem. The considered unit is arriving at 10:00 and scheduled to depart at 15:00. It cannot be assigned to the first track since the remaining space is insufficient. The fourth track is also not feasible since an existing unit has to leave (14:00) before the considered unit, thus blocking the innermost unit on the track. The second and third tracks are both feasible assignments because they have sufficient remaining space and no unit is blocked. Two exact solution methods for the DPP, based on mathematical models, are considered in Chapter 7. In addition, a framework for integrating the this problem with the rolling stock schedule is considered, where the DPP is a submodule in the framework.

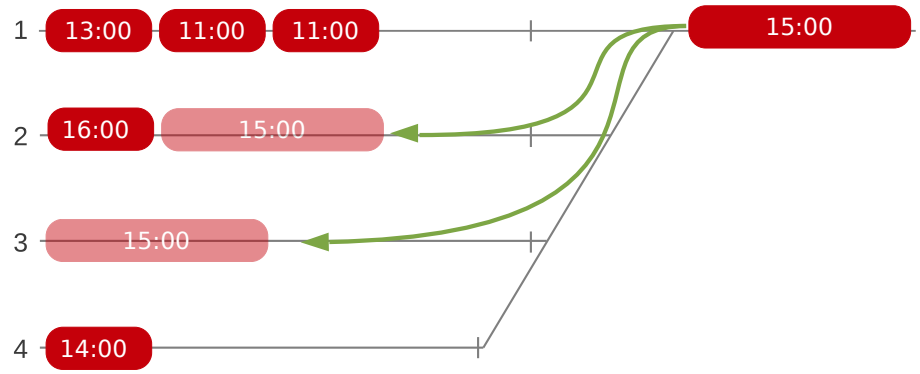


Figure 2.6: An example of a depot parking decision. Train units are represented as red boxes. The departure times of all units are shown. At some time, say 10:00, a unit is arriving and requires a track assignment.

In general a DPP instance contains a set of arriving and departing events at some depot with a set of available tracks. Most tracks are last-in first-out (LIFO) tracks which means that they are accessible from one side as shown in Figure 2.6. The problem can be extended to include tracks that are accessible from both ends, so-called *free* tracks [13].

Although it is assumed that units enter and exit the depot in isolation, this is not always the case in practice. Train compositions with more than one unit may arrive or wish to leave the shunting area. Handling them as a whole is beneficial as it may reduce the number of necessary coupling or decoupling operations. Kroon et al. [20] present a Mixed Integer Program (MIP) formulation that considers parking compositions as a whole.

The DPP assumes that an itinerary is given for each rolling stock unit, i.e., it is known when each physical rolling stock unit is arriving and departing. Although possible, internal shunting movements are usually not considered; a plan that does not require moving units between arrivals and departures is clearly preferred. In tactical planning a rolling stock schedule is based on the flow of unit types, and not individual units. Units of the same type are considered interchangeable, and therefore the list of events in the DPP is not tied to any specific unit. Consequently multiple feasible matchings may exist that map individual units to departures.

The Departure Matching Problem (DMP) considers matching arrivals and departure at some station. It is assumed that units are parked in an associated shunting area until they are scheduled to depart again. In a simple variant of the problem, all departures must be assigned to a

specific requested unit type. This assignment problem is easy to model and solve using a MIP formulation.

Integrating matching and parking poses a more difficult problem, and it is considered in Chapter 6, where multiple solution methods are compared. Solving both problems in isolation (in sequence) can in some instances fail to produce a feasible solution. Although many feasible matchings exist for the DMP, many of these matchings may be infeasible to park when considering the subsequent DPP. In Chapter 6 we propose solution methods to solve the integrated problem. In addition, parking a subset of units on platforms is considered. Some of the last rolling stock units to arrive at a station may, without much trouble, stay on a platform until the next day given that they will depart early the following day. This extension is useful and even necessary in some cases when depot capacity is scarce.

In Chapter 8 the DMP is studied with additional complicating constraints. The considered problem is a sub-problem in the "ROADEF Challenge 2014 - Trains don't vanish!!!" [2]. The matching problem is mainly complicated by the fact that rolling stock units have maintenance constraints - a decision must be made when to perform maintenance. Furthermore, a subset of the specified departures are *linked* to an arrival, which means that same physical unit will return as an arrival at a later point in time. In Chapter 8, a column generation approach and a MIP solution method are considered for solving this sub-problem. In turn, the column generation approach is used as a component in the solution method to solve the full problem of the challenge. For a description of a solution method for the full problem see Appendix A.

2.6 Crew Planning

Train drivers must be assigned to all train services as specified by the timetable. Essentially, the Crew Planning Problem (CPP) stipulates that schedules must be assigned to each available train driver such that all train services are covered. One of the main objectives is to minimize operational costs. A review of the problem and solution methods is given by Caprara et al. [9].

A graph representation of the problem is very similar to the one of rolling stock, see Figure 2.3. Similar to rolling stock units, crew members can get on and off a train and perform different subsets of train services. In contrast, only one driver is needed to perform a train trip, and they can easily move (deadhead) from one position to another for example by taking a train, bus or taxi. The set of tasks (or duties) performed by a crew member must however adhere to a number of constraints. A non-exhaustive list is given below:

- Sufficient time is required in order to switch between two tasks, including logging in/out of a train and walking from one cockpit to another.
- A crew schedule must have sufficient time for on-duty breaks, such as meal-breaks.
- Resting periods between working hours must be respected such as breaks between two days of work and weekends.
- Union rules may impose a certain level of route variety, e.g., driving the same segment (in the same direction) more than once (during the same duty) is not acceptable in general.
- Drivers are possibly not certified to drive all types of rolling stock, and a plan must therefore respect the individual crew limitations.

Multiple methods have been proposed to solve the CPP in literature [9], also in a real-time rescheduling context [23, 22]. Several variants of the problem have been studied as differences exist among railway operators, e.g., across different countries. Several railway operators have now adopted automated decision support system for handling crew planning [9].

2.7 Freight Hump Yard Planning

Freight and passenger railway planning problems have many similarities. In many cases both types of transport share the same infrastructure and are subject to the same type of constraints. Freight trains are operated less frequently and are usually longer and heavier than passenger trains. The timetable for freight trains can be considered more flexible as they are not bound by a published service contract but rather an agreement between companies. A timetable route is still required to allow passage through the shared infrastructure. It is of paramount importance that the allocated route through the infrastructure network is smooth, i.e., without requiring full stops or many drastic changes in velocity. In addition to the inability to rapidly change speed, it is very energy consuming to accelerate a heavy train, see Section 2.8.

Demand for freight transportation exists between multiple origins and destinations in a network. However, it is impractical to offer transporting service between all such end-points. Hump yards (classification yards or marshaling yards) are consolidation points for freight traffic. Inbound freight trains are split by processing them over a so-called *hump*. The hump is located on a small hill, thus allowing gravity to pull individual railcars (a block) through a series of switches into parallel tracks in a classification yard (the bowl). In contrast to the shunting areas mentioned in Section 2.5, these tracks are processed in a first-in first-out (FIFO) order. Finally, sequences of railcars are pulled from a classification track and coupled onto outbound trains. The net effect is a split, sort and merge process. Each railcar is bound to a specific destination, and it is vital to sort the railcars such that the scheduled outbound trains can depart with as many cars as possible. The goal is to minimize the average dwell time (compare to throughput time in manufacturing) of the cars; time spent in the hump yard corresponds to revenue lost.

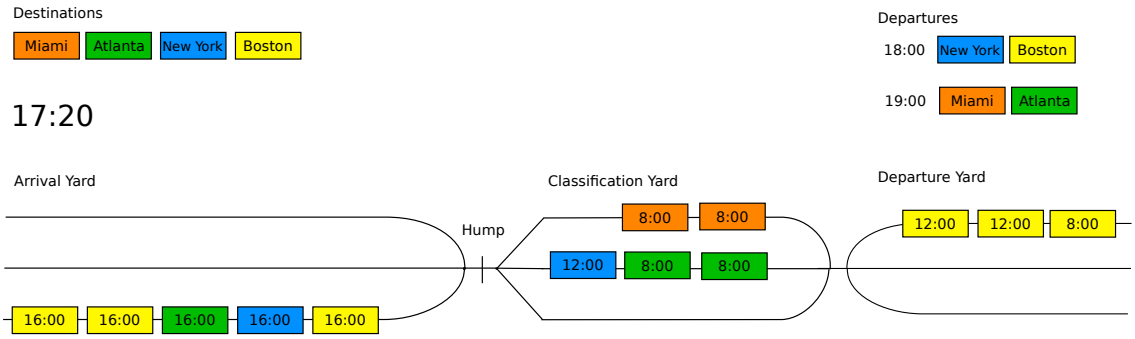


Figure 2.7: An example of a hump yard instance.

Figure 2.7 shows an example of a hump yard instance. Four different destinations and two departures are included. Rail cars are illustrated using boxes, and the arrival time of the cars are shown inside the box. Three inbound trains are present (8:00, 12:00, 16:00) and the example shows a possible execution at time 17:20. The two first inbound trains have been processed over the hump, and the yellow cars (previously on the bottommost classification track) have been pulled to build the first part of the 18:00 outbound departure train. The departure at 18:00 is now close and observe that a few cars from the 16:00 train will not meet this departure, if only three pullout operations can be performed every hour.

The RAS Problem Solving Competition [1] introduced the described Railroad Hump Yard Block-to-Track Assignment Problem (HYBA). Several assumption are made in order to make the problem manageable, but the essential planning problem remains. In the HYBA, the set of arrival trains, for the next period of days is known. It is part of the problem to decide in which order to process the trains. A daily recurring departure schedule is also given. A departure can only consists of an ordered set of rail car destinations (i.e. a standing order). One inbound train can be processed by the hump at a time releasing up to two cars every minute. It is assumed that inbound trains

cannot be partially humped, once moved to the hump they must be fully processed. In the bowl the tracks naturally enforce FIFO restrictions since a car cannot overtake other cars on the same track. A few hours in advance the outbound trains may be built using a limited set of pullout train engines simultaneously. Naturally, the engines cannot work on the same tracks, thus a minimum separation time is enforced. Two pullout tasks from the same track are prohibited if there is insufficient separation time. In addition, two pullout tasks cannot be assigned to the same outbound train unless sufficient time is separating them. The minimum separation time is an interesting candidate for increasing robustness. Increasing or inserting buffer time for performing pullouts permits the personnel opportunities to catch up, if they become delayed. However, this also decreases the number of possible pullout opportunities thus potentially inflicting revenue losses.

In general, inbound trains are not required to be processed atomically on the hump. Although inconvenient, a partial humping may be beneficial. In literature [6], re-humping is considered a viable option. Such an extension allows the planner to re-hump cars in a classification track by pulling them back to the hump. Granted, this is inconvenient but it allows more flexibility. If the average inbound train is fragmented (in terms of rail car destinations), and if the number of destinations is higher than the available classification tracks, then re-humping may be very beneficial. Finally, freight trains are not bound by a public timetable such as passenger trains. Delaying departures is therefore possible. Depending on the state of the classification yard, it may be beneficial to delay an outbound train if it results in more rail cars being able to depart. Forcing the rail cars to wait for the next slot (24 hours) is bad due to the increase in dwell time but also because tracks on the classification yard are blocked during that period.

In Chapter 9 a solution method for the problem [1] based on decompositions and matheuristics is presented. Real-life instances with more than 3 million feet of railcars are benchmarked. A number of what-if scenarios are also performed in order to identify possible bottlenecks in the system. A solution method to the HYBA therefore also serves as an interesting tool for investigating possible infrastructure investments.

The problem instances considered in Chapter 9 contain planning horizons of multiple weeks. Naturally, it is too optimistic to assume that nothing will change in the course of this period. The actual demand is likely to deviate from the given forecast, thus changing the composition of the inbound (arriving) trains. Furthermore, the execution of a plan may deviate from the plan. A robust plan should, to some degree, be resilient in the face of such change by handling small changes gracefully. The execution of a plan can be made more robust by increasing the headway or buffer times to perform certain tasks. However, it is difficult to construct a plan that is able to recover from a change in the order of arriving rail cars. An online rescheduling tool accommodates these robustness issues by re-planning the problem with updated information, thus obtaining a reactive robustness. The solution method presented in Chapter 9 finds high quality solutions within a few minutes and is therefore eligible as a rescheduling solution method.

2.8 Energy-Efficient Driving

When planning timetables it is necessary to have run-time estimates between all consecutive stops on a train service. The minimal time required to move the train between the full stops can be determined analytically by modeling train dynamics using equations of motion. The fastest speed profile (requiring the least amount of time) is simply determined by following the speed limits as closely as possible, i.e., exercise full power when driving less than the speed limit, and brake just in time before the speed restriction is lowered. The run-time estimate used in the timetable planning process is therefore set to the minimal time required plus some amount of buffer time.

Admittedly, the time available for performing a trip journey is usually greater than the minimal required time. The fastest speed profile is not energy-efficient as savings can be made by performing

coasting or cruising at a lower velocities. A coasting phase can usually be performed before the braking phase in order to save energy, but at the cost of time. This results in less energy-loss as no power is used in the coasting phase. Further, a lower cruising speed (i.e. holding speed) can reduce energy consumption as train resistances are expressed as a cubic function of the velocity.

A more energy-efficient speed profile can be achieved at the cost of using additional time. In the coasting phase time is lost due to a lower velocity in the coasted segment. In the cruising phase, a lower velocity naturally results in a later arrival. Essentially the goal is to find the best tradeoff between increasing coasting segments and lower cruising speeds. Note, in some cases the train can regenerate power while braking thus complicating the optimization problem further. Figure 2.8 shows an example of different speed profiles for the same train journey. Examples of alternative cruising speeds and coasting options are illustrated. Multiple speed restrictions may exist that require the train to brake in advance.

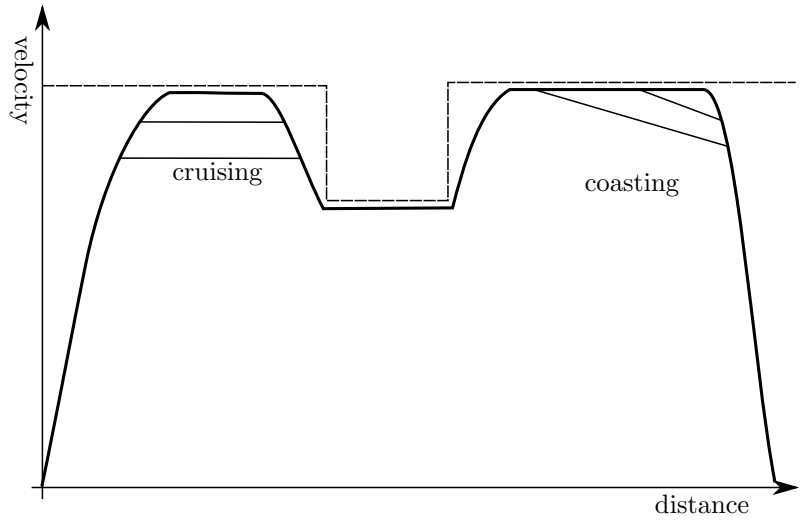


Figure 2.8: The plot shows examples of feasible speed profiles respecting the speed-limits drawn using dashed lines. The bold path illustrates an aggressive strategy following the speed-limits as closely as possible by using full acceleration and braking, thus arriving as soon as possible. Alternative and more energy efficient profiles can be achieved, at the expense of a longer trip time, by lower cruise speed or by performing coasting as shown. The available travel time limits the number of feasible profiles.

Theoretical research work has been established by Howlett and Pudney [18], and in [17] it is shown that an optimal driving strategy, for a region with one speed-limit, consists of a single power-hold-coast-brake (i.e, accelerate-cruise-coast-brake). The basic assumption is that no steep hills exist such that holding speed is always possible. The optimal solution can be found using a binary search strategy [16]. In Figure 2.8 we observe that when multiple speed-limits exist, this strategy does not hold if time is critical. However, multiple power-hold-coast-brake stages can be performed. The optimal solution is, however, no longer simple to find.

In addition to determining energy efficient speed profiles, a solution method can also improve the robustness in railway operation. Using an on-board DAS, a train driver is guided to follow a speed profile that helps the driver to arrive on time (in addition to saving energy). Without disruptions or real-time changes such speed profiles can be determined offline. Further robustness can be achieved by ensuring that the profile arrives at intermediate checkpoints at certain times. Adopting precomputed (offline) speed profiles faces one fundamental and important issue, namely uncertainty. Some changes occur on a short notice such as temporary speed restriction, route allocations and timings. Precomputing profiles for all possible scenarios is therefore impractical. A solution method that is able to find a solution within few seconds can therefore provide reactive robustness. In addition to being able to cope with short notice changes, such fast methods can

also adapt to the imperfect driving of the train driver. A driver can easily fall behind or go ahead when following a given speed profile, thus requiring an updated profile.

In Chapter 10 an elegant solution method based on Dynamic Programming is proposed to optimize the train speed profile problem. By considering the intersections made between selected partial profiles, a distance by velocity graph is constructed. The arcs in the graph represent driving choices, e.g., acceleration, cruising, coasting and braking. A label setting algorithm is presented to solve the resulting resource constrained shortest path problem. The solution method is eligible for real-time speed profile rescheduling, thus providing reactive robustness. It is able to find solutions withing one second on average but within two seconds in the worst case.

Chapter 3

Conclusion

Planning railway operations is far from trivial. Even though this vast task in practice is decomposed into multiple interdependent optimization problems it remains non-trivial since the individual problems can be hard to solve. A number of these strongly interrelated railway optimization problems have been studied during this PhD project. Reactive robustness is challenging as it asks for lower run-time requirements while basically solving a recovery problem identical to or very similar to the original planning problem. Integrating strongly related problems is very attractive as it potentially avoid infeasibility in later planning stages and potentially produces better plans. However, integrating problems usually only adds to the difficulty of solving the problems. Due to the combinatorial complexity and the element of uncertainty, the railway industry still calls for further research in order to extend, integrate and find faster and better solution methods for planning problems in the railway sector.

In this thesis a number of integrated approaches have been studied. The traditional sequential processing of interdependent railway problems can, as argued earlier, results in infeasibility or in a low solution quality. Integrating the interrelated problems therefore aims at producing plans of better quality and avoiding infeasibility at the cost of an increased problem complexity. In Chapter 6 depot matching and parking has been integrated and several different solution methods are benchmarked. A greedy heuristic proves to be efficient at finding feasible solutions while a number of other exact methodologies prove to be less useful when considering large problem instances with many different rolling stock types. In Chapter 8 and Appendix A matching, parking, maintenance and routing has been integrated thereby obtaining a holistic view of a large railway station. The integrated problem proved to be difficult to solve as none of the participants of the corresponding challenge were able find plans that serviced all arrivals or departures for the given *artificial* instances. An integration of rolling stock and depot parking is studied in Chapter 7. The study confirms that this integration is nontrivial as it entails joining depot stack ordering constraints with inventory flow management in the rolling stock circulation. A solution framework based on a cut-back routine is proposed to solve the problem.

It has been argued that the computational time requirements for any solution method is essential when considering reactive robustness. An eligible approach for rolling stock rescheduling is considered in Chapter 4. The benchmark shows that real-life data instances from DSB S-tog can be solved in reasonable time for real-time support. This approach is extended to include composition rules in Chapter 5 thereby enforcing a more strict (and robust) operation. Benchmarks on real-life data instances from DSB S-tog and Nederlandse Spoorwegen (NS) show low computational requirements. In Chapter 6 it is shown for the integrated depot problem that heuristic methods are able to find solutions very promptly while exact solution methods have difficulties to do the same for large instances. Train speed profiles can be computed in 1-2 seconds using the Dynamic Programming (DP) method presented in Chapter 10. This low computational time makes the

method a suitable candidate for decision support if, for example, the driver falls behind or ahead of his planned schedule, or if real-time changes such as temporary speed restrictions make the planned profile infeasible.

Extensions to existing railway problems are commonly studied in the literature, e.g., in order to strengthen existing approaches or to investigate new aspects. Different railway operators face distinct variations of the same base problems as the goals, bottlenecks or business rules may differ. In Chapters 4 and 5 a new rolling stock model has been proposed that is able to handle unit-specific restrictions such as maintenance constraints. The benchmarks show that the proposed solution method is efficient. In Chapter 6 a number of different solution methods for matching and parking rolling stock units are compared in a benchmark. A platform extension is included in order to model overnight parking at DSB S-tog. An extended variant of the depot matching problem is studied in Chapter 8. In contrast to previous variants, rolling stock units are subject to different types of maintenance constraints, which plays an important part as train departures will later return as arrivals. This problem is proven to be $\mathcal{NP}$-hard in the strong sense, and both proposed exact methods are unable to solve the challenge instances to optimality. A passage points extension is considered in the solution method for finding train speed profiles in Chapter 10. The extensions allows planners to set a time window on specific locations, e.g. for signals, to ensure that the train passes this point without interference, e.g. a red signal to stop.

Main contributions

The main contributions of this thesis are the contributions of the papers (presented in individual chapters) produced during the PhD project. The contribution takes the form of problem definitions, theoretical considerations, problem extensions, solution method development and benchmarking. The contribution of the individual units of work are listed below:

Chapter 4: A Branch-and-Price methodology for solving the Rolling Stock Rescheduling Problem is presented and benchmarked. Unlike most other similar work, aggregated depot capacity constraints are included. A novel unit-based rolling stock model is presented that can be solved using delayed column generation. The solution framework is tested on recent real-life datasets provided by DSB S-tog, including realistic disrupted cases generated by applying existing contingency plans. A test with maintenance constraints is also performed.

Chapter 5: A path based MIP formulation for the Rolling Stock Rescheduling Problem is proposed. In this formulation the order of units in train compositions is taken into account. A column generation solution approach (which is an extension of the work in Chapter 5) is proposed. Dynamic row generation is adopted for a significant speedup in run-time of the solution approach. The solution approach is compared to a state-of-the-art flow-based MIP approach, and both are benchmarked on real-life instances of the Danish State Railways and Nederlandse Spoorwegen.

Chapter 6: Several new methods for the Depot and Matching Problem are proposed: A constraint program, a column generation framework, a two-phase decomposition method and a greedy heuristic approach. These methods and a reference MIP model are benchmarked on realistic datasets provided by the Danish State Railways and Nederlandse Spoorwegen. The work also considers the possibility of parking rolling stock units on platform tracks at the end of the day in order to deal with scarce depot capacity. The second place of the "RAS Student Paper Award" was awarded for this work.

Chapter 7: Two variants of an integrated framework for rolling stock and depot plans are presented and tested. The first variant is an exact approach that solves the problem within few minutes. Further, a short comparison of methods for solving parking problem is performed that shows that the column generation approach [13] in fact is outperformed by a reference

MIP. A counter example is given that highlights the heuristic nature of a previously proposed optimal method in [13] for generating the optimal Linear Program (LP) solution to the so-called Track Assignment Problem (TAP).

Chapter 8: A definition for the Departure Matching Problem (DMP) is given; a distinct, self-contained sub-problem of the Rolling Stock Unit Management on Railway Sites [2]. The problem is proven to be $\mathcal{NP}$-hard in the strong sense by reduction from the 0-1 Multiple Knapsack Problem. Two optimization-based approaches are presented for solving instances of the DMP: a MIP solved using a commercial solver, and an equivalent MIP mathematical model that is solved using column generation. The given instances (by the ROADEF Challenge [2]) are not be solved to optimality but solutions are obtained in a heuristic setting using time limits. The second junior (PhD) prize was awarded for the work (Appendix A) at the IFORS2014 conference in Barcelona, where the column generation approach as used as a sub-component.

Chapter 9: A matheuristic framework is presented that solves hump yard processing, classification yard sorting and departure building in an integrated manner. The benchmarks shows that, within minutes, the method is able to produce high quality solutions for the INFORMS RAS competition problem instances [1]. A set of *what-if* scenarios are investigated in order to identify bottlenecks or possible future investments. The first prize award was received for this work at the INFORMS 2014 conference in San Francisco.

Chapter 10: A novel DP methodology for optimizing the Train Speed Profile Problem is proposed and benchmarked. The method does not rely on advanced software packages or mathematical solvers. In addition to gradient forces and speed-restrictions the solution method can also handle passage points. To the best of my knowledge a solution method for (multiple) passage points has not been proposed in literature before. Compared to previous approaches space nor time is uniformly discretized, but an event-based space-velocity graph is computed. The choices represented in the underlying graph are optimized using a label setting algorithm that minimizes energy usage subject to the time constraint. A large benchmark of real-life instances shows that the method is sufficiently fast for real-time decision support.

Chapter 11: A novel MIP based approach and several matheuristic variations are proposed to solve the problem of synchronizing train departures in order to minimize energy cost. The problem is introduced in a research challenge by the Friedrich-Alexander University Erlangen-Nürnberg [3]. The problem instances can be solved to a low optimality gap within a few hours using the MIP based approach. The matheuristic approaches fail to outperform the MIP approach with time limits. An honorable mention (and second place) was awarded for the high quality and *excellent work* submitted during the challenge.

Practical Impact

Security and reliability are two governing qualities of the railway operation, which means that changes happen slowly. Proven and extensively tested technology is preferred and might be accepted after comprehensive trials of testing and approval. Arguably, the practical impact of this PhD project can only be projected. The following paragraphs include some of the potential implications of the work.

In Chapters 4 and 5 a column generation framework is proposed for scheduling rolling stock that, opposed to previous approaches, models individual rolling stock units more explicitly. This is a step into the right direction since it allows a better handling of unit-specific constraints such as maintenance requirements. The benchmarks performed show that the presented methods are sufficiently fast for resolving disruptions. With appropriate modifications, such a method can be used as decision-support tools. Currently, automated crew rescheduling is being tested at the Danish State Railways (DSB) - rolling stock rescheduling seems like an obvious next candidate

to implement. In addition, the presented method framework can, without much difficulty, be transformed into a heuristic method, e.g, using a local search strategy for composing a rolling stock schedule using a pool of possible units schedules.

In Chapter 6 methods for planning depot movements are benchmarked, while in Chapter 7 an integration of rolling stock scheduling and depot matching and parking is considered. With appropriate extensions, the depot planning and integration with rolling stock scheduling seems useful for the DSB, where optimized rolling stock schedules often are infeasible due to the lack of a feasible depot plan. Furthermore, platform parking is addressed, which is a necessity due to scarce depot capacities today in the suburban railway network in Copenhagen.

An efficient matheuristic is proposed in Chapter 9 for producing detailed rail-car plans in large hump yards. In this work rail-car humping, classification (sorting) and departure building is considered in one framework, thus all steps are to some degree integrated. The relatively low run-time requirements suggest that this framework could be used in operation to plan efficient processing of incoming freight trains. The potential of building new yards, or upgrading existing ones, can also be estimated using the same method.

Trains are heavy and consume a significant amount of energy when traveling between stations on a journey. In Chapter 10 an elegant and flexible solution framework is presented that is able to outperform a heuristic method provided by an industrial partner. Using the solution method the amount of consumed energy can be reduced significantly, especially compared to the energy consumed by an energy-inattentive driver. With the addition of modeling passage points, this solution method extends existing work in literature, which can be used to improve overall robustness. Finally, with short computational run-times the framework can be adopted for real-time decision support, which facilitates reactive robustness.

Future Research Directions

The railway industry presents many challenges which cannot be considered *solved* yet. This industry is also cautious to absorb new research as reliability is of paramount importance. In order to be accepted by the railway partners more research is required to demonstrate the importance and benefits of using optimization models and tools. Closing the gap between academia and industry is here important as it would help to build confidence in the research. Some interesting future research directions are listed below:

- Scheduling rolling stock in short-term or real-time requires a high level of detail, in contrast to tactical or strategic schedules. Including explicit routes for rolling stock units has been confirmed by the DSB to be a move in the right direction. In this thesis the individual units have been included in a rolling stock model. In future research, it is also interesting to consider the routes for extensions that exploit this modeling framework.
- The freight hump yard problem presented in Chapter 9 is an interesting problem where humping, sorting and departure building is solved in an integrated framework. Investigating the potential of extending the presented approach to a strategic tool could prove very useful when determining future investments. The solution method does not produce plans with re-humping or partial humping. Investing the potential of re-humping or partial humping would be interesting.
- The problem studied in the ROADEF Challenge 2014 [2] (see Chapter 8 and Appendix A) considers many interesting aspects of routing trains in a (large) station. Further research and application has been endorsed by analysts at the DSB. The central station in Copenhagen would be an interesting case study.
- An online tool for calculating energy efficient train speed profiles has already been motivated. Further improvement and extensions of the work presented in Chapter 10 would

prove interesting. Improving the quality of the generated solutions is one interesting topic to pursue, but extending the solution framework to handle steep hills, discrete speed control and regenerative braking is also an interesting direction.

Finally, more interesting future research directions are presented and discussed in the individual chapters.

Bibliography

[1] Ras problem solving competition. `https://www.informs.org/Community/RAS/Problem-Solving-Competition/2014-RAS-Problem-Solving-Competition`, 2014. Accessed: 2015-07-06.

[2] Roadef/euro challenge. `http://challenge.roadef.org/2014/en/index.php`, 2014. Accessed: 2015-07-03.

[3] Fau open research challenge - discrete optimization: Energy-efficient train timetables. `https://openresearchchallenge.org/`, 2015. Accessed: 2015-07-30.

[4] Ahuja, Jian Liu, Orlin, Sharma, and Shughart. Solving real-life locomotive-scheduling problems. *Transp. Sci. (USA)*, 39(4):503–517, 2005.

[5] Arianna Alfieri, Rutger Groot, Leo Kroon, and Alexander Schrijver. Efficient circulation of railway rolling stock. *Transportation Science*, 40(3):378–391, 2006.

[6] Markus Bohlin, Florian Dahms, Holger Flier, and Sara Gestrelius. Optimal freight train classification using column generation. In Daniel Delling and Leo Liberti, editors, *ATMOS*, volume 25 of *OASICS*, pages 10–22. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[7] Valentina Cacchiani and Paolo Toth. Nominal and robust train timetabling problems. *European Journal of Operational Research*, 219(3):727–737, 2012.

[8] Luis Cadarso and Ángel Marín. Robust routing of rapid transit rolling stock. *Public Transport*, 2(1-2):51–68, 2010.

[9] A. Caprara, L.G. Kroon, M. Monaci, M. Peeters, and P. Toth. Passenger railway optimization. In C. Barnhart and G. Laporte, editors, *Handbooks in Operations Research and Management Science*, volume 14, chapter 3, pages 129–187. Elsevier, 2007.

[10] Jean-François Cordeau, François Soumis, and Jacques Desrosiers. Simultaneous assignment of locomotives and cars to passenger trains. *Oper. Res.*, 49(4):531–548, July 2001.

[11] Stacy C Davis, Susan W Diegel, and Robert G Boundy. Transportation energy data book. 2009.

[12] Pieter-Jan Fioole, Leo Kroon, Gábor Maróti, and Alexander Schrijver. A rolling stock circulation model for combining and splitting of passenger trains. *European Journal of Operational Research*, 174(2):1281–1297, 2006.

[13] Freling, Lentink, Kroon, and Huisman. Shunting of passenger train units in a railway station. *Transp. Sci. (USA)*, 39(2):261–272, 2005.

[14] Julie Jespersen Groth, Daniel Potthoff, Jens Clausen, Dennis Huisman, Leo Kroon, Gábor Maróti, and Morten Nyhave Nielsen. Disruption management in passenger railway transportation. *Robust and Online Large-Scale Optimization*, pages 399–421, 2009.

[15] Jørgen Thorlund Haahr, Thomas Stidsen, and Martin Zachariasen. Heuristic methods for single link shared backup path protection. *Journal of Heuristics*, 20(5):539–560, 2014.

[16] Ingo Arne Hansen and Jörn Pachl, editors. *Railway Timetabling & Operations*. Eurailpress, 2008.

[17] Phil Howlett. The optimal control of a train. *Annals of Operations Research*, 98(1-4):65–87, 2000.

[18] Philip G Howlett and Peter J Pudney. *Energy-efficient train control*. Springer, 1995.

[19] Leo Kroon and Dennis Huisman. Algorithmic support for railway disruption management. In Jo A.E.E. Nunen, Paul Huijbregts, and Piet Rietveld, editors, *Transitions Towards Sustainable Mobility*, pages 193–210. Springer Berlin Heidelberg, 2011.

[20] Leo G. Kroon, Ramon M. Lentink, and Alexander Schrijver. Shunting of passenger train units: An integrated approach. *Transp Sci*, 42(4):436–449, 2008.

[21] Lars Kjær Nielsen, Leo Kroon, and Gábor Maróti. A rolling horizon approach for disruption management of railway rolling stock. *European Journal of Operational Research*, 220(2):496–509, 2012.

[22] Daniel Potthoff, Dennis Huisman, Daniel Potthoff, Dennis Huisman, and Guy Desaulniers. Column generation with dynamic duty selection for railway crew rescheduling. *Transp Sci*, 44(4):493–505, 2010.

[23] Natalia J. Rezanova and David M. Ryan. The train driver recovery problem—a set partitioning based model and solution method. *Computers & Operations Research*, 37(5):845 – 856, 2010.

[24] Alexander Schrijver. Minimum circulation of railway stock. *CWI QUARTERLY*, 6:205–217, 1993.

[25] Joris Wagenaar and LG Kroon. Maintenance in railway rolling stock rescheduling for passenger railways. *ERIM Report Series Reference No. ERS-2015-002-LIS*, 2015.

Part II

Rolling Stock and Depot Planning

Chapter 4

A Branch-and-Price Framework for Railway Rolling Stock Scheduling

Jørgen Thorlund Haahr* Richard Martin Lusby* Jesper Larsen* David Pisinger*

*Department of Management Engineering, Technical University of Denmark,
Produktionstorvet, Building 426, DK-2800 Kgs. Lyngby, Denmark
jhaa@dtu.dk, rmlu@dtu.dk, jesla@dtu.dk, pisinger@man.dtu.dk

[1] **Abstract** Scheduling or rescheduling rolling stock is a passenger railway optimization problem. In current practice this is typically optimized manually despite the high complexity and high runtime requirements of the task. In this paper we propose a path-based mathematical formulation that is solved using column generation in a complete Branch-and-Price framework. In contrast to flow-based approaches our formulation is more easily extended to handle certain families of constraints, such as train unit maintenance restrictions. We benchmark the framework against real-life instances provided by the suburban railway operator in Copenhagen (DSB S-tog). In combination with a lower bound method we show that near-optimal solutions can be found within a few seconds during a disruption. In addition, we show that the framework is also able to find solutions within a few minutes for non-disturbed timetables.

4.1 Introduction

Rolling stock scheduling (or circulation) is a non-trivial task which must be addressed by railway operators on a regular basis. It is often difficult to just find a feasible solution, i.e., an assignment of physical train units such that all train services are covered, depot capacities are respected, end-of-day depot balances are correct and such that several other business rules are satisfied. Several feasible schedules may exist and it is important to find a good solution as rolling stock is one of the major expenses for a railway company. Rolling stock schedules are usually made months in advance by planners. Dispatchers might, a few days prior to operation, slightly adjust these before assigning individual train units to the schedules. Rolling stock scheduling has been the subject of much research over the last two decades, and significant advances have been made in this area. An approach by Schrijver [32] considers a model for determining the minimum number of units needed for a single train line. Fioole et al. [19] propose a richer model that includes train

[1]Revision submitted in *Transportation Science*, July 2015

compositions, demand shortage, shunting movements and train splitting. More recently, Nielsen et al. [27] adopt a version of this model for disruption management. The majority of models and solution methods focus on offline planning and consider approaches that assign anonymous train flow, in contrast to routing individual train units.

Inevitably, unplanned events that disturb the pre-planned operation occur on a daily basis. Depending on its severity, a disruption may render the planned rolling stock schedule infeasible. Examples of large disruptions include: passenger induced delays, rolling stock breakdown, signalling problems, and infrastructure damage or failure. Such disruptions occur multiple times per year, and the situation requires a new schedule to be found immediately. The impact of a disruption may be hard to comprehend for dispatchers and high quality recovery solutions may be very hard to find. If not handled well, the effects of a disruption can easily cascade and result in even more problems. Real-time rolling stock rescheduling is an emerging area of research that has received relatively limited attention in the literature [27, 21]. In recent years there has been a growing interest in this field; however, the topic still remains a challenge.

Our main contribution is the presentation and benchmark of a Branch-and-Price (B&P) methodology for solving the Rolling Stock Rescheduling Problem (RSRP). Column generation is an interesting technique that has proven to work well with many problems. It is already state of the art for railway crew scheduling [29, 30]; but, applications in rolling stock rescheduling are limited [7, 28]. We present a unit-based rolling stock model that can be solved using delayed column generation. To the best of our knowledge such a methodology for the RSRP has not been published before. A unit-based model presents advantages, e.g. in its ability to model maintenance constraints. Further, we present and benchmark our framework on recent real-life datasets provided by DSB S-tog (railway operator in the greater Copenhagen area). Using these datasets, we also generate realistic disrupted cases by applying existing emergency guidelines. In a separate benchmark we test maintenance constraints. This is a valuable extension for practitioners responsible for tactical and operational scheduling. The capacity at individual station depots is scarce and is often a limiting factor for rolling stock plans at DSB S-tog. Unlike most other work, we therefore include depot capacity constraints, ensuring that the total available track is never exceeded.

The aim of this paper is to present a new methodology. Some details and modeling aspects in disruption management are thus considered out of scope. We refer the reader to the recent paper [27], which presents interesting disruption modeling extensions such as penalizing deviations from planned activities and embedding a solver in a rolling horizon framework. Although omitted in this work, such extensions are applicable. Naturally, passenger demand changes during disruptions and depends on factors such as the disrupted location and the available capacity in the new rescheduled plan. It is a non-trivial task to come up with new reliable demands for the timetable. This is the topic of [13]. We note that similar extension could be applied to our proposed framework. The demand of the normal timetable will be adopted for the disrupted ones.

This paper is structured as follows. A description of the problem as well as a literature review are given in Section 4.2. Section 4.3 outlines a mathematical formulation of the RSRP, while Section 4.4 presents our B&P framework. Section 4.5 describes the test instances and shows the experimental results. Finally, conclusions are discussed in Section 4.6

4.2 Problem Description

To the public a timetable is a chart stating departure and arrival times of all trains in the system. The timetable consists of a set of *lines*, where each line is a set of *train trips* (train services) which connect the same origin and destination pair and contain the same subset of intermediate stations. A limited number of trips on the same line can start or end at some intermediate station that is different to the origin and destination. Every train trip is an individual journey from one origin

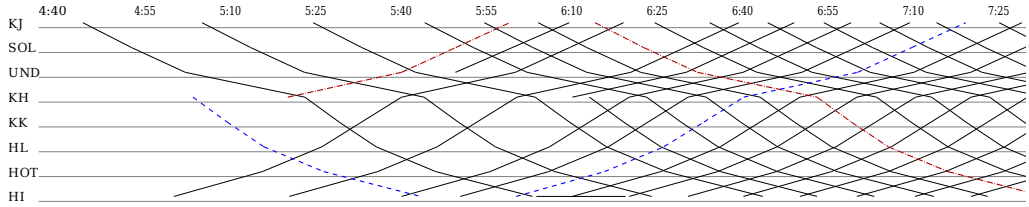


Figure 4.1: An example of a timetable for a single line with trip services going between two end-stations. Two trip-sequences have been highlighted for illustrative purposes.

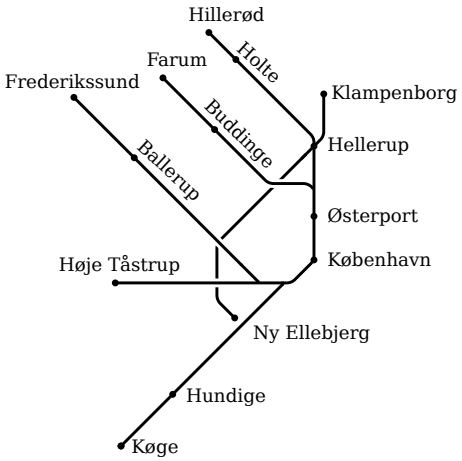


Figure 4.2: The S-tog railway infrastructure network. The depot- and end-stations are highlighted.

station to a destination station, and determines the departure times at every station from the origin to the target. Figure 4.1 shows a time-space diagram of a single line containing multiple trips. A *trip sequence* denotes a continuous series of trips which can be performed by the same physical train composition. That is, the same composition can be assigned to each trip in the trip sequence without going into or out of the depot or require any shunting activity. Trip sequences typically span a whole day and stay within a single line. In some cases, a trip sequence alternates between lines in the network. Each line consists of a set of trip sequences and every trip, except *deadheading* trips, is assigned to a single trip sequence. Deadheading trips are non-revenue train movements that are used to re-position rolling stock.

The operators need a detailed plan that specifies which rolling stock is used between every two consecutive stops in the timetable; but, such a fine grained level of detail is not needed. We introduce *subtrips* (or train segments) that provide an appropriate level of detail. A trip is partitioned into subtrips where each subtrip define segments between key stations, i.e., terminal stations, turning stations and stations where trains can be coupled or decoupled. Figure 4.2 illustrates the key stations in the S-tog network. During disruptions a few additional key stations can appear since an infrastructure blockage may require trains to turn around at other stations.

We use train unit and rolling stock interchangeably; this refers to the smallest inseparable vehicle we consider, i.e. a set of rail cars which, for practical reasons, cannot be split or combined on a day-to-day basis. In our study we only have one family of units, consisting of two different types; this means that every unit can be coupled to any other unit available, resulting in a longer train with an increased seat capacity. Additionally, all train drivers can drive either type, and all trains are self-propelled, i.e., do not need a locomotive for pulling. The important type characteristics are: the number of seats, the physical train length, and the operational cost.

Units can be coupled together to form train compositions, where a composition is a set of units in a specific order. Figure 4.3 shows possible compositions. In this work the compositions are implicitly

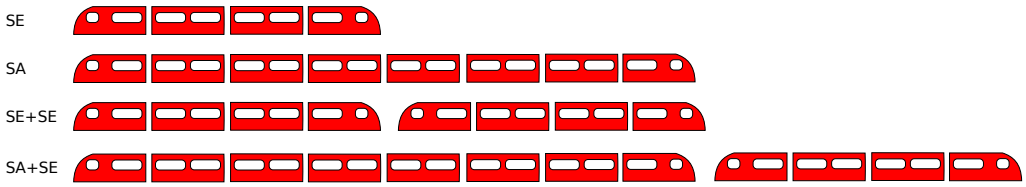


Figure 4.3: Examples of train compositions.

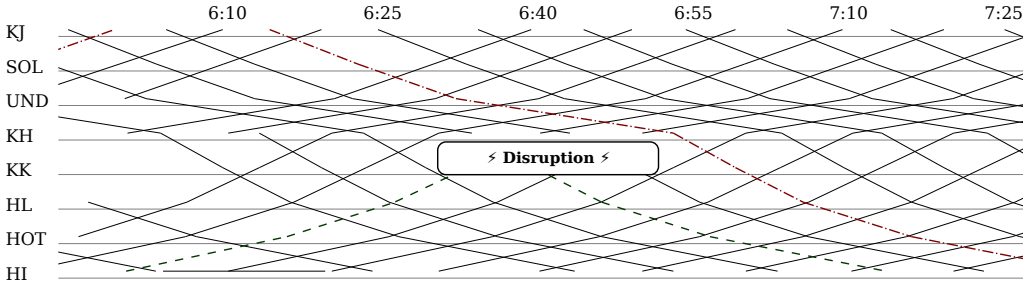


Figure 4.4: A disrupted timetable due to infrastructure blockage between station KH and KK. The updated timetable resolves the situation by turning the train services at station KH and KK.

determined by the individual assignment of train units to subtrips. We argue that the composition plays a smaller role in our study for two reasons. Firstly, compositions on trips typically range from one to two units in size at DSB S-tog. This means that there are no ordering conflicts when coupling or decoupling units. An uncommon composition with three units can occur; however, in this composition all units must be of the smaller type. Secondly, during a disruption, the ordering of individual units is not a governing priority.

In this paper, coupling or decoupling represents a shunting operation in line with the definition of several existing papers (see e.g. [19]). A coupling is performed whenever a unit is moved from a depot and onto a train service. Likewise, a decoupling is performed whenever a unit leaves a train service and enters a depot. In both cases, a change in train composition is made. Recall that a trip sequence can be serviced by the same composition without requiring any shunting activity; as long as a unit performs subtrips of the same trip sequence in succession, no coupling or decoupling is needed. We only inflict a coupling (or decoupling) penalty when train enters (or leaves) a trip sequence. Consequently, a subtrip can have one successor and one predecessor subtrip in the same trip sequence. We assume that a subtrip has at most one successor or predecessor, in contrast to a time-table that includes combining or splitting of train services.

The RSRP entails assigning train units to every subtrip in a given time horizon. The assignment is subject to a number of natural constraints such as fleet capacity, train unit location, and depot capacity. The problem is similar and comparable to the multicommodity flow problem. Note that we also adopt an equivalent definition which we find more natural from a train-unit perspective: A rolling stock schedule is a chronological list of tasks performed by every available train unit.

4.2.1 Disruption Management

Trains do unfortunately not always arrive and depart on time. This is due to unplanned events such as accidents, human mistakes, infrastructure failures, rolling stock breakdowns, and weather conditions. In this paper we consider events that cannot be absorbed by network buffers and will result in a disrupted situation on the network. In such cases, network operators must make changes to the timetable in order to avoid a total breakdown.

Any changes to the timetable may render the rolling stock schedules and other planned activities

infeasible. Figure 4.4 gives an example of a disrupted line due to a blockage in both directions between two stations. Given a revised timetable from the infrastructure owner, a new, candidate rolling stock schedule must be presented by the rolling stock dispatchers within a short time. In reality, the timetable and rolling stock schedule are not determined completely independently. The infrastructure owner cannot expect to carry out a timetable that is infeasible with respect to rolling stock, and they are also not interested in cancelling more train departures than necessary. Thus, when rescheduling rolling stock, it makes sense to integrate timetable decisions to some degree. In this paper we assume that we can assign no rolling stock units to any subtrip, at a high cost. This implies that the corresponding subtrip is cancelled. Note that we assume there is no penalty for cancelling deadheading subtrips as these do not affect the public timetable.

During a disruption the situation can be quite chaotic; it must be resolved as quickly as possible, giving all actors little time to reschedule plans. In our case study the infrastructure owners have exclusive control over all train activities, while train operators have a supporting role. The infrastructure owners respond to a disruption quickly by making train service cancellations or alterations. Predefined emergency plans exist and are used as guides for handling the disruption. Their immediate goal is to avoid delay propagation and further escalation. Depending on the time of the day and the location of the disruption, the available time before a decision must be made ranges from 3 to 20 minutes. We refer to Groth et al. [20] for an in-depth description of the disruption management process. A temporary timetable is formed and the operators must try to reassign train units according to the new disrupted timetable. This is a non-trivial task. The new plans may be far from optimal since the issues are resolved sequentially by degree of severity using rules of thumb, while only considering the immediate future. Train units of cancelled train services are often put into the nearest depot without considering the long-term consequences. After the disruption is resolved the original timetable is restored by the infrastructure owners, and (in our case study) the railway operator has, by contract, 60 minutes to recover the rolling stock (and all other) schedules. It may, however, be impossible to cover all trains at this point in time due to the effects of the disruption. The network operator must thus be able to find new plans quickly both during and after the disruption. This motivates the use of a computer-aided decision support system.

Computer-aided decision support systems are already used by DSB S-tog during the different planning phases for determining the rolling stock schedules. The system is not eligible for real-time rescheduling. For detailed real-time planning the current positions of the units and any individual unit restrictions must be adhered to. Important unit properties such as time or mileage limits – due to maintenance requirements – should be respected. Other less critical unit-specific properties are even interesting to consider in a planning context, e.g., rolling stock with bike space or units displaying commercials.

4.2.2 Objectives

The primary objective of the RSRP is feasibility; however, if several solutions exist, then some schedules will be more attractive than others. Therefore, we include the following secondary objectives: the number of trip cancellations, robustness notions, seat shortages, end-of-day depot balances and operational costs.

Trip cancellations are the most expensive component as these contribute to the reliability performance of the railway operator. The real price-function is complex as a fine is issued if the reliability percentage falls below a certain contracted threshold at year-end. The size of the fine depends on the deviation of the reliability.

The concept of robustness is often ambiguous and can include several things. In this paper we do not decide on arrival or departure times; the robustness can only be affected by the choice of assigned rolling stock units. Couplings and decouplings are performed many times daily. Although necessary, these operations reduce overall robustness. A (de)coupling can easily take more time

than expected due to manual interaction and unforeseen technical difficulties. In addition, it also requires coordination with a second train or a depot driver. Minimizing the number of couplings is important. Performing the couplings outside critical time periods and reserving enough time for (de)couplings can also be considered a robustness indicator (see Cardaso et al. [11] for more consideration on robustness). In this paper we impose a minimum time required to perform (de)couplings.

The railway operator is obliged to satisfy the actual demand on a trip by supplying enough seats to cover some percentage of the passengers. Therefore, it is not only necessary to cover a trip but also to assign units, i.e. a composition, that collectively provide enough seats.

The end-of-day depot balance is important for continued operation the next day since a certain number of trains must be available at each depot the following morning. If the balance is off, unplanned deadheading trips will be necessary to execute the plan the next day of operation. Deadheading train units is undesirable due to the expense of running empty rolling stock units and allocating additional drivers to move the trains.

The operational cost of driving a train mainly consists of power consumption, wear and tear, infrastructure allocation, cleaning, and driver related expenses. The first two are linear with respect to the distance travelled and the other costs are more or less fixed depending on whether the trip has been cancelled or not. The infrastructure is already reserved and it has already been decided how many drivers are working on the day of operation. Cleaning costs are constant as the unit utilization is close to 100% every day.

The number of cancellations is usually (in literature and at DSB S-tog) the most important objective and is therefore given a cost that is significantly higher than the other objectives. The seat and operational objectives are conflicting and a non-trivial trade-off is sought. The network operator is obliged to meet the passenger demand, but not at all costs, e.g., the operator would not couple a rolling stock unit to a train service only to provide few additional passenger seats. In the benchmarks we investigate this trade-off and try to find a sensible balance. Note, that this effectively undermines the importance of finding optimal solutions in contrast to merely good solutions. Finally, the end-of-day balance is relevant when handling disruptions, as this may obscure the fleet position at the end of operations. A sufficient number of train units must be present at specific depots for the following day. A mismatch is corrected by performing deadheading trips, e.g., during the night.

In testing our solution method we do not introduce a more complicated objective for disruptions. Examples of this can be found in see [27] and [30], where a separate objective minimizing deviation from the original plan is used. In contrast to crew scheduling, identical rolling stock units can be interchanged without substantial complications w.r.t. the trip coverage, seat shortage and operational expense. We penalize the end-of-day deviations, but do not penalize missed or unplanned shunting operations (in addition to the normal cost) during disruptions. Changing plans is understandably a nuisance that requires communication and coordination. However, with the recent advancements in mobile technology we believe that this is a diminishing issue.

4.2.3 Literature

Most literature related to passenger railway optimization (from an operator perspective) can be categorized by the problem addressed, e.g., timetable planning, rolling stock scheduling, crew scheduling, or shunting. In this section we limit our discussion to the somewhat more scarce literature on rolling stock scheduling with an emphasis on real-time disruption management. For an overview of different railway optimization problems the reader is referred to Caprara et al. [14], while Kroon and Huisman [25] and Jespersen-Groth [22] provide overviews in a disruption management context. We consider self propelled trains units that do not require a powered locomotive for pulling. Several papers exist that consider the locomotive and carriages problem, see

e.g. [1, 15, 16, 17, 33, 26].

The rolling stock problem has been studied in a variety of different settings using different mathematical models and targeted time horizons. A good reference for the rolling stock scheduling problem is the one used by the Dutch railway company Nederlandse Spoorwegen (NS), described by Fioole et al. [19]. The authors present the problem of assigning train unit compositions to every departure. The main goal is to cover every train trip while respecting several constraints such as fleet capacity, composition change restrictions and allowing trains to be combined and split.

The strategic or tactical rolling stock problem has been studied in various forms over the past two decades. The goal is to determine the fleet size or to find a feasible rolling stock schedule. An early approach by Schrijver [32] minimizes the number of required trains to satisfy passenger demand. This approach does not consider train compositions, composition changes, depot parking nor maintenance. A small data-instance from NS is presented and solved within seconds.

Alfieri et al. [3] consider a richer rolling stock model that considers the compositions of train arrivals and departures. The model assumes a single line and one family of compatible train units. A generic workday on a line from NS is solved within 1-2 hours. We essentially consider the same rolling stock problem; however, we also model individual rolling stock trajectories and restrict the number of parked trains to the capacity, i.e. total track length in meters, at each station depot. We do not include constraints limiting the composition changes.

Peeters and Kroon [28] study a similar setting spanning multiple lines using one family of train unit types. A B&P method which outperforms a commercial Mixed Integer Program (MIP) solver is presented. Computation times are less than a minute for the larger data instance consisting of three lines. A few what-if scenarios are considered that quantify the effect of changing the fleet composition (and platform lengths) and allowing shunting at a new station. Compared to our work, the authors present a different column generation formulation of Alfieri et al. [3] model; sequences of composition changes are modelled as variables, while we model unit trajectories as variables.

Fioole et al. [19] study an extension of the same model that allows combining and splitting trains. The authors consider subset of lines from NS and solve the problem as a MIP using a commercial solver. In addition, a heuristic based on the Linear Program (LP) relaxation is used. Good quality solutions are found within hours. The solution model was used as a supportive tool for generating the Dutch timetable for 2005.

Cadarso et al [12] consider a formulation similar to Fioole et al. [19] that additionally tries to optimize the robustness of the rolling stock schedule while penalizing high densities of standing passengers. Deadheading train units is allowed in order to increase capacity. Robustness is modelled by penalizing the total number of coupling operations, and by giving shunting operations during rush hours a special penalty. They consider two data instances: a single day of a single line from RENFE with 4 depot stations and 320 train services, and two lines with 9 depot stations and 400 train services. Optimal solutions are generated within minutes using a commercial MIP solver.

A two-stage optimization model is presented by Cacchiani et al. [9] for solving the rolling stock scheduling problem with a set of generated failure scenarios. The model is similar to the one presented by Fioole et al. [19], but without minimum shunting time, and the combining and splitting of trains. A full MIP is presented and solved heuristically using Benders Decomposition. The data instance is a one day of single line from NS with up to 400 trips and 28 failure scenarios. The solution method is able to find good solutions within 2 minutes.

Cacchiani et al. [10] consider a train-unit assignment problem with self-propelled rolling stock. It is solved using a heuristic based on a so-called customary column generation methodology. Solely the number of units used is minimized, leading to a handful of theorems and one lower bound scheme. Unlike our approach, [10] do not include costs for shunting, seat shortage and driven

mileage. Furthermore, demand is modelled as a hard constraint and shunting capacity constraints at station depots are not included. An extension considering maintenance is proposed stating that all rolling stock units must contain at least one maintenance opportunity during a weekly schedule.

Ziarati et al. [33] present a locomotive assignment problem which is structurally similar to the problem considered in this paper. The authors propose a column generation solution method that relies on heuristic fixing in a Branch-and-Bound (B&B) methodology. Sufficient engine capacity (number of locomotives, horsepower and tonnage) must be assigned to train segments in order to pull the preassigned train cars. Like our work, units must be assigned to cover the train segments in order to meet a demand and total mileage is penalized. In contrast, we propose a full branching scheme and explicitly model train (de)coupling activities. Other minor modeling differences exist such as allowing cancellations, end-of-day balance deviations.

Cordeau et al. [17] consider the locomotive and cars assignment problem at VIA Rail Canada. In contrast to our work, this problem involves assigning rail cars to satisfy demand as well as sufficient traction engines to pull the cars. The authors benchmark a column generation framework using heuristic branching and fixing rules to obtain solutions of good quality. In comparison, we model seat shortages and cancellations as soft constraints using penalties and do not need constraints to ensure sufficient traction. We include coupling costs directly in our framework whereas [17] resolve these *switching operations* in a second phase. Depot capacity constraints are modelled, and sufficient time in a single planning cycle to perform maintenance is ensured. Subsequent work by Cordeau et al. [16] presents a benders decomposition framework that improves computational times of especially the larger instances.

Several studies consider a short-term rolling stock problem aiming at rescheduling a few days before the day of operation. Compared to a long-term model such approaches have to deal with a greater level of detail and cannot assume a clean-slate approach as the availability of the different resources is fixed. Budai et al. [8] consider a rebalancing problem where the goal is to minimize inventory end-of-day deviations from the original plan. Two very fast heuristics methods, which aim to improve existing schedules are presented. The authors prove that the rebalancing problem is NP-hard. Lingaya et al. [26] study the problem of rescheduling locomotives and carriages at VIA Rail Canada. Ben-Khedher et al. [6] present a schedule planning system and a tactical capacity-adjustment system for increasing revenue at SNCF.

The RSRP rolling stock problem is a relatively new area of research. Online approaches focus on railway disruption management. This is not possible for offline approaches partly because of the strict runtime requirements, but also because of different goals, restrictions or level of detail.

Nielsen et al. [27] propose a setting and framework that is similar to the work of this paper. A rolling stock scheduling framework is presented that uses time horizons to overcome disruption uncertainty and runtime difficulty. The optimization model is similar to the model presented by Fioole et al. [19], but without the possibility of splitting and combining trains. The main goal is to minimize the number of cancelled trains, end-of-day off-balances and changes to the original shunting plan. The end-of-day train unit inventory balances are heuristically determined through the horizons. Every horizon is solved using a commercial MIP solver.

Cadarso et al. [13] present an approach which integrates rolling stock and dynamic passenger demand. The model may cancel train services on predetermined lines, insert emergency train services and determine the direction of one-way tracks in case of one-way blocked segment. Passenger demand is updated iteratively after solving the rolling stock model until the demand stabilizes. The solution method is benchmarked on a 2 hour disruption case study provided by RENFE consisting of 5 lines, 10 depot stations and 760 train services. Solutions are found within a few minutes. The case study does not reveal significant changes in passenger demand after the first iteration.

Jespersen-Groth [20] studied a variety of DSB S-tog related problems. One of which focused on the reinsertion of train lines once a disruption is over. The model considers the distribution

$\mathcal{S}$	Set of stations
$\mathcal{D}$	Set of stations that have an associated depot
$\mathcal{T}$	Set of subtrips
$\mathcal{U}$	Set of train unit types
$\mathcal{A}$	Set of all arrival events across all stations in $\mathcal{S}$
$\mathcal{P}_d^u$	Set of all possible paths for unit type u starting at station d

Table 4.1: List of sets

of available trains and the time required for drivers to reach them. The model is solved as a MIP which minimizes the time of the last inserted train. The test instances are solved within 30 seconds.

4.3 Model

We now define the RSRP and notation more formally before presenting the proposed mathematical model. To distinguish between parameters and decision variables, the latter always appear in bold type. We denote by $\mathcal{S}$ the set of key stations, which is an appropriate aggregation of all stations in the railway infrastructure. The set $\mathcal{D} \subseteq \mathcal{S}$ defines the set of stations with an associated depot. Furthermore, we let $\mathcal{U}$ be the set of all train unit types. We assume that all unit types are compatible and self-propelled, i.e., all units can be coupled together and no locomotive is required to pull the train. Finally, we let $\mathcal{T}$ be the set of all timetabled subtrips that have to be serviced. A subtrip $t \in \mathcal{T}$ departs at a certain time from a source station and arrives at a certain time at a target station. For convenience, we define $\mathcal{A}$ to be the set of all subtrips arrivals. We define $depot(a)$ as the shorthand for the depot-station of the arrival event.

The RSRP is the problem of finding an assignment of all available units to subtrips. The assignment of each unit must respect the inherent temporal and spatial constraints, i.e., a unit can only be at one place at the same point in time and can only be assigned to a subtrip if it is located at the departure station of the subtrip prior to the subtrip's departure time. A unit entering or leaving a trip sequence must have reserved enough time to perform the (de)coupling before being assigned or reassigned to any other activity. Finally, the depot parking space is limited which means that only a certain total length of train units can stay parked at depot-stations any point in time. Consequently, a unit cannot perform a subtrip and then park if the destination depot is full.

We consider the following objectives in the mathematical model: train trip coverage, seat shortage, end-of-day balance, (de)couplings and operational cost. Trip coverage is directly related to trip cancellations, and it is our primary concern to minimize the number of cancellations. If a subtrip in $t \in \mathcal{T}$ is uncovered, the trip is considered cancelled. Matching the demand is also important; thus, the seat shortage is minimized. The number of available seats depends on how many units service a subtrip. The difference between available seats and the demand defines the shortage. However, if there is a surplus of seats then the shortage is zero. The number of (de)couplings affect the robustness of the solution, and hence should be minimized. We approximate the number of (de)couplings by counting the number of times a unit enters and leaves a trip-sequence. This is not an accurate number of required shunting operations; however, it measures how many times we change the composition of a trip-sequence which is directly related to robustness. We do not consider schedule deviations in the objective function in our approach; the model can, however, be extended to do so with the addition of more constraints and variables.

Instead of presenting a compact flow-based formulation of the rolling stock rescheduling problem we present a path based model with an exponential number of path variables. We let $\mathcal{P}_d^u$ denote the set of all paths for unit type $u \in \mathcal{U}$ starting at depot-station $d \in \mathcal{D}$. A path defines a feasible

$\boldsymbol{\lambda_p}$	Integer variable which determines how many units travel on path p
$\boldsymbol{y_t}$	Binary slack variable which takes value 1 iff subtrip t is covered
$\boldsymbol{z_t}$	Integer slack variable determining the seat shortage on train t
$\boldsymbol{w_d^u}$	Integer slack variable determining end-of-day balance shortage for unit type u at depot d

Table 4.2: List of Variables

c_1^t	Cost of cancelling a subtrip t
c_2	Cost of one seat shortage per kilometre
c_3	Cost of one end-of-day balance shortage
c_4	Cost of one coupling or decoupling
c_5^u	Operational cost per kilometre for unit type u
s_u	Number of seats on train unit u
$demand_t$	Seat demand for subtrip t
$length_t$	Maximum train composition length allowed for subtrip t
$inventory_d^u$	Number of units of type u starting at depot d
eod_d^u	Target end-of-day balance for train unit u at depot d
$track_d$	Combined length of tracks at depot d

Table 4.3: List of parameters

κ_t	Number of kilometres on subtrip t
ξ_p	Number of couplings and decouplings imposed by path p
κ_p	Number of travel kilometres accumulated on path p
α_p^t	Binary coefficient which takes the value 1 iff path p visits subtrip t
β_p^d	Binary coefficient which takes the value 1 iff path p terminates at depot d
$\gamma_p^{d,a}$	Binary coefficient which takes the value 1 iff path p is staying at depot d on or before the arrival of subtrip $a \in \mathcal{A}$

Table 4.4: List of Coefficients

trajectory of a rolling stock unit and indicates a set of subtrips which are serviced by a unit that is following the path. We only consider paths that can be feasibly performed w.r.t time and spacial constraints, e.g., a path cannot contain two subtrips that overlap in time nor can it perform two tasks in sequence if the arrival station of the first is not equal to the departure station of the second. The cost of a path depends on the sum of accumulated mileage and the number of couplings and decouplings performed. These shunting operations are implicit as a unit has to be parked whenever the pause between two subtrips is too long. The rescheduling problem consists of assigning exactly one path to each vehicle.

The mathematical model contains four sets of decision variables, one of which are binary and three of which are integer. First, $\boldsymbol{\lambda_p} \in \mathbb{Z}_0^+$ controls how many units of a certain type use path $p \in \mathcal{P}$, where $\mathcal{P} := \bigcup_{u \in \mathcal{U}, d \in \mathcal{D}} \mathcal{P}_d^u$. Second, $\boldsymbol{y_t} \in \{0,1\}$ is a slack variable that determines whether a subtrip $t \in \mathcal{T}$ is cancelled. If set to value 1 subtrip t is cancelled, otherwise the variable has value 0 and t is not cancelled. Third, $\boldsymbol{z_t} \in \mathbb{Z}_0^+$ is a slack variable that counts the total seat shortage on subtrip $t \in \mathcal{T}$. Finally, $\boldsymbol{w_d^u} \in \mathbb{Z}_0^+$ denotes the number of units of type $u \in \mathcal{U}$ that are missing in depot $d \in \mathcal{D}$ from the scheduled end-of-day balance. The model is presented below.

$$\text{Minimize: } \sum_{t \in \mathcal{T}} \left(c_1^t \boldsymbol{y_t} + c_2^t \kappa_t \boldsymbol{z_t} \right) \tag{4.1}$$

$$+ \sum_{u \in \mathcal{U}} \sum_{d \in \mathcal{D}} c_3^{d,u} \boldsymbol{w_d^u} \tag{4.2}$$

$$+ \sum_{u \in \mathcal{U}} \sum_{d \in \mathcal{D}} \sum_{p \in \mathcal{P}_d^u} (\xi_p c_4 + \kappa_p c_5^u) \boldsymbol{\lambda_p} \tag{4.3}$$

The notation is summarised in Tables 4.1, 4.2, 4.3 and 4.4. The objective function is a weighted average of five components. In (4.1) the cost of cancellations and seat-shortage are added. In (4.2) the end-of-day balance shortage cost is added. Finally, in (4.3) the (de)coupling and mileage costs are added.

$$\sum_{p \in \mathcal{P}_d^u} \boldsymbol{\lambda_p} = inventory_d^u \qquad (\pi_d^u) \qquad \forall u \in \mathcal{U}, d \in \mathcal{D} \tag{4.4}$$

$$\sum_{p \in \mathcal{P}} \alpha_p^t \boldsymbol{\lambda_p} \geq 1 - \boldsymbol{y_t} \qquad (\mu_t) \qquad \forall t \in \mathcal{T} \tag{4.5}$$

$$\sum_{u \in \mathcal{U}} s_u \sum_{d \in \mathcal{D}} \sum_{p \in \mathcal{P}_d^u} \alpha_p^t \boldsymbol{\lambda_p} \geq demand_t - \boldsymbol{z_t} \qquad (\delta_t) \qquad \forall t \in \mathcal{T} \tag{4.6}$$

$$\sum_{p \in \mathcal{P}} \beta_p^d \boldsymbol{\lambda_p} \geq eod_d^u - \boldsymbol{w_d^u} \qquad (\omega_d^u) \qquad \forall u \in \mathcal{U}, d \in \mathcal{D} \tag{4.7}$$

$$\sum_{u \in \mathcal{U}} l_u \sum_{d \in \mathcal{D}} \sum_{p \in \mathcal{P}_d^u} \alpha_p^t \boldsymbol{\lambda_p} \leq length_t \qquad (\phi_t) \qquad \forall t \in \mathcal{T} \tag{4.8}$$

$$\sum_{u \in \mathcal{U}} l_u \sum_{d \in \mathcal{D}} \sum_{p \in \mathcal{P}_d^u} \gamma_p^{d,a} \boldsymbol{\lambda_p} \leq track_{depot(a)} \qquad (\nu_a) \qquad \forall a \in \mathcal{A} \tag{4.9}$$

$$\boldsymbol{y_t} \in \{0,1\} \quad \boldsymbol{\lambda_p}, \boldsymbol{z_t}, \boldsymbol{w_d^u} \in \mathbb{Z}_0^+ \tag{4.10}$$

Constraints (4.4) ensure that the number of paths for a specific unit type leaving a depot corresponds to the number of units of that type available at the depot. The second set of constraints (4.5) ensures that $\boldsymbol{y_t}$ is set to 1 if subtrip t is not covered by any path. Constraints (4.6) ensure that the lack of seats is captured by $\boldsymbol{z_t}$, while Constraints (4.7) make sure that the end-of-day balance shortage is captured in $\boldsymbol{w_d^u}$ for every depot d and unit type u. Note, that we could as easily model and penalize surpluses; however, assuming the total end-of-day balances equals

the fleet size one rolling stock shortage necessarily results in one surplus. Thus, it is unnecessary to penalize both. The maximum train composition length is restricted by (4.8), according to the platform lengths at the subtrip's endpoints. Constraints (4.9) enforce the requirement that the capacity at any depot d must be respected by every arrival event. If the track capacity at every arrival is satisfied up until the last arrival, then the schedule respects the depot capacities throughout the day. Paths can go in and out of multiple depots multiple times during the horizon, and multiplying $\gamma_p^{d,a}$ by the number of units currently residing in the depot provides the total track usage. Finally, Constraints (4.10) restrict the domains of the variables. Note that the slack variables are naturally integral if the path variables are integral. The dual values associated with the constraints of the model's LP relaxation are given in parentheses.

Naturally, we do not propose to solve the full model. Instead we propose using delayed column generation. Such path based models have been proven to improve runtime drastically for a number of routing problems using a delayed column generation approach [18]. Column generation resembles the primal simplex method; i.e., at every iteration a variable (i.e. column) with negative reduced cost is selected as a candidate to enter the basis. As the number of variables is exponential, we rely on a so-called *subproblem* (or pricing problem) to find such variables. The subproblem itself is an optimization problem that needs to be solved at every iteration in order to identify variables with negative reduced cost, or to prove that none exist. Note, that we do not achieve a tighter LP relaxation for the RSRP, compared to an arc or flow based formulation. A tighter LP relaxation is, however, obtained when we extend the problem with maintenance restrictions, and thereby require the subproblem to find resource constrained shortest paths in contrast to simple shortest paths. The proposed formulation can be improved by replacing Constraints (4.5)-(4.7) with hard constraints, thus reducing the solution space of the problem. However, we do not pursue this as it is uncertain during disruptions whether this is possible, and furthermore we want to enable investigation of different trade-offs. In addition, we propose that the integrality of the formulation can be improved if Constraints (4.5)-(4.6) are removed and constraints, trying to achieve a *desired* composition, are used instead. In some cases operators already know which compositions they would like to (or cannot) assign. As is, Constraints (4.6) encourage fractional covers.

4.3.1 Rescheduling Extension

Considering a single horizon in the middle of a whole day presents some challenges which requires some minor changes to the model. Essentially these changes also allow the framework to work using a rolling horizon, c.f. [27].

Trains are running continuously and some decisions made a few minutes into the past cannot be undone, e.g., if a train unit was assigned to a departure happening before the horizon begins, then the train will continue this trip until it arrives at the next station. Likewise, any train unit assigned to a departure before the horizon ends must perform the trip into the following horizon.

In the proposed model every train unit starts at a depot and is available from the beginning of the planning period. However, inside a horizon some trains will only become available some time after the start of the horizon, due to being en route when the horizon begins. This implies that some units will be unable to cover the first subtrips. To model this, we partition the sets of unit types available at each station $s \in \mathcal{S}$ further, based on a set of n_s discrete event times $\tau_s = \{\theta_1, \theta_2, \ldots, \theta_{n_s}\}$. Associated with each time $\theta \in \tau_s$ is a known number of available units of each type $u \in \mathcal{U}$. For the sake of completeness, we assume this number is $inventory_s^{u,\theta}$, and note that, for all $u \in \mathcal{U}$, $inventory_s^u = \sum_{i=1}^{n_s} inventory_s^{u,\theta_i}$. Constraints (4.4) hence become:

$$\sum_{p \in \mathcal{P}_s^{u,\theta}} \boldsymbol{\lambda_p} = inventory_s^{u,\theta} \qquad \forall u \in \mathcal{U}, s \in \mathcal{S}, \theta \in \tau_s, \tag{4.11}$$

where $\mathcal{P}_s^{u,\theta}$ contains all paths for unit type $u \in \mathcal{U}$ originating from station $s \in \mathcal{S}$ after time $\theta \in \tau_s$. The other constraints must now sum over all these paths and not just for all depots and unit types

alone. We note that this modification does not affect the structure of the subproblem and does not increase the number of subproblems to solve.

Another complicating issue that arises is how to determine the number of couplings. Units arriving on a subtrip from the previous horizon may have been routed to a platform (in order to continue on some trip sequence) or into the depot for parking. Since the arrival times of these units occur after the beginning of the horizon, the previous decisions can be altered. Thus, such arriving units are denoted *undecided*, and the model decides whether it should be parked or if it will continue on the immediate, succeeding subtrip. As such, the price of decoupling must be paid in the current horizon as it was undecided whether the subtrip assignment would lead to a decoupling once the unit arrived.

Couplings and decouplings are further complicated if we impose minimum (de)coupling times. In this case the model might have decided to park a unit arriving in the last horizon, and this unit will not be available in the current horizon until a certain time has elapsed. In contrast to before the unit is now available from the depot only, and can not take a subtrip from the platform immediately. This issue can be modelled the same way as the undecided units by adding a new similar set of constraints and limiting the possible paths emerging from the depot.

4.4 Branch-and-Price Framework

In this section we propose a B&P framework for solving the RSRP. This well known technique for solving large-scale integer programs combines column generation with B&B [5]. Column generation is typically preferred when the mathematical model contains a vast number of variables. To implement column generation, one relaxes all integrality restrictions and decomposes the problem into a *master problem* and one or more independent *subproblems*. The master problem contains only a subset of the variables for the full problem, and is typically referred to as the Restricted Relaxed Master Problem (RRMP). The subproblems are optimization problems responsible for generating variables (i.e. columns) that are not included in the RRMP, but which have the potential to decrease the RRMP's objective function value. More specifically, each subproblem utilizes the dual information from an optimal solution to the RRMP, and attempts to identify negative reduced cost columns that can be added to the RRMP. Column generation refers to the iterative procedure between master and subproblems that must be performed. Incorporating this into a B&B setting in which columns are generated (or "priced") at each node of the B&B tree gives rise to the B&P terminology. This section focuses on specific details concerning our B&P approach. Figure 4.5 illustrates the steps in the framework. First, a master model is generated with initial *feasibility* columns, i.e., a set of variables are initially inserted such that a feasible solution exists for the LP relaxation. The RRMP is solved and the dual values are used to find variables with negative reduced cost by the subproblem. Whenever new variables are found the RRMP must be resolved, otherwise the RRMP is optimal. A B&B node is finished when an optimal solution to the RRMP is found. If the solution is feasible and respects the integral constraints of the original master problem, then the upper bound is updated otherwise new branching nodes are created. The framework terminate once all nodes are processed.

4.4.1 Master Problem

The RRMP is obtained from Model (4.1)-(4.10) by replacing constraints (4.10) with

$$\boldsymbol{y_t} \in [0,1], \quad \boldsymbol{\lambda_p}, \boldsymbol{z_t}, \boldsymbol{w_d^u} \in \mathbb{R}_0^+, \tag{4.12}$$

and restricting the set of path variables for each depot and unit type pair (d,u) to a subset $\mathbb{P}_u^d \subset \mathcal{P}_d^u$. Each column defines a legal path through the rail network for a unit of type u originating from

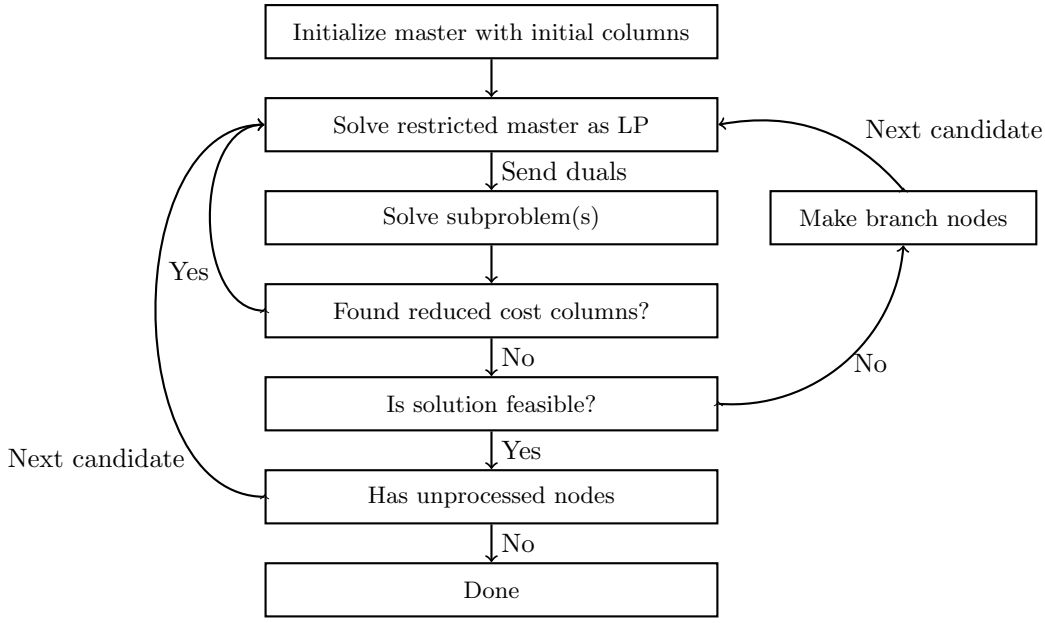


Figure 4.5: Flow diagram for the proposed Branch-And-Price framework

depot d Recall that for this relaxed version, the dual variables π_d^u, μ_t, δ_t, ω_d^u, ϕ_t, and ν_a are associated with Constraints (4.4), (4.5), (4.6), (4.7), (4.8), and (4.9), respectively.

4.4.2 Subproblem

The role of a subproblem is to identify one or more negative reduced cost columns. As indicated above, this can be viewed as the pricing step in the conventional simplex algorithm, with the exception that all variables are not stored and priced explicitly. Variables with negative reduced cost for the RRMP are returned by the subproblem(s). The variables correspond to paths for particular unit types. We formulate the problem of finding such paths as a Shortest Path Problem on an acyclic time-space network (see e.g. [33] and [17]). A subproblem is defined for each unit type. This implies that, in the absence of individual restrictions, any unit of the given type can perform the path. Note that the underlying network structure of each subproblem is identical since we assume any unit can perform any subtrip; however, some parameters of each subproblem must be modified to reflect the individual unit types. In what follows we present a more detailed description of the time-space network. To ease the understanding, the network is introduced in stages.

Not described in this paper is the extension of the subproblem into a Shortest Path Problem with Resource Constraints (SPPRC) (see e.g. [18] and [24]). We perform some benchmarks with unit maintenance restrictions which we solve using a label-setting algorithm. We do not describe this algorithm in detail as the implementation is straightforward considering we only have one resource and the underlying graph is a directed acyclic graph. We refer the reader to [23] for details on solving such problems.

Underlying Network

To generate a path one must identify a *feasible* sequence of subtrips starting and ending at particular depots. Modelling all such possibilities for a given planning period can be achieved via a time-space network. In such a network, every node corresponds to a particular *event*. Here an

event can refer to a departure, a passthrough, or an arrival. A departure event refers to the start of a trip sequence, while an arrival event is associated with the end of a trip sequence. A passthrough event represents a midway stop between two subtrips - one incoming and one outgoing subtrip is associated to each passthrough event. Any subtrip indicates the movement of a train between two particular stations at a certain time and thus can be introduced as an edge in the time-space network.

Associated with each subtrip edge is a passenger demand and a subtrip operating cost $c_5^u \kappa_t$. Recall that the operating cost depends on the unit type under consideration. Furthermore, each subtrip is associated with a constraint from (4.5), a constraint from (4.6), and a constraint from (4.8). Thus, the reduced cost contribution (or weight) of each such edge is $\rho_t := c_5^u \kappa_t - \mu_t - s_u \delta_t - l_u \phi_t$. Additionally, edges which indicate a depot arrival are adjusted by $-l_u \nu_a$ consumption of depot track capacity (see Constraints (4.9)). By adding a super source as well as a super sink node, connecting the former to all departure events and the latter to all arrival events, we obtain an acyclic, time-space network, in which all trips within the planning period are implicitly represented by paths. Note that the weight of edges emanating the origin is $c_4 - \pi_d^u$, while edges to the sink have weight $c_4 - \omega_d^u$. The (de)coupling cost c_4 is indicates that the unit enters and exits a trip sequence.

Figure 4.6 below illustrates such a time-space network for a simplified example with three stations, two of which are depot stations (denoted d_1 and d_2). The example contains two trip sequences (each having three trips), and there are 12 unique subtrips. All nodes corresponding to station events are ordered in time, and appear on the same horizontal level. The network is labelled assuming we are pricing a unit of type u. An example path from depot-station d_1 to depot-station d_2 is given, along with a selection of example edge weights. Note that this network does not allow decoupling at intermediate stations. Here a and a' denote the two arrival subtrips at the depot.

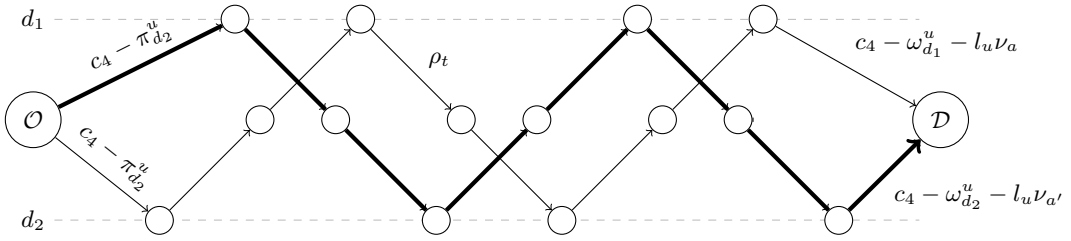


Figure 4.6: Simplified example of the acyclic time-space network. This figure illustrates the subtrips of two trip sequences. The blue path is an example of a possible unit path. The current network does not capture depot details.

While this time-space network includes all subtrips in the planning horizon, it is inflexible in the sense that it does not permit a unit to be decoupled from a trip sequence; since each node in the graph has degree at most two, each path must consist of a sequence of consecutive subtrips. This is not ideal; it may be preferable to allow units to follow only part of a trip-sequence. To provide such flexibility, we duplicate all nodes for station events occurring at depot-stations, creating a *platform event* node as well as a *depot event* node. This creates more possibilities for a unit, given its location at the time of the respective station event. A platform event stipulates that the unit type is at the platform and must perform the outgoing subtrip, while a depot event indicates that the unit type is positioned in the depot and cannot perform the outgoing subtrip.

To allow a unit type to remain parked at a depot-station we introduce edges connecting consecutive depot events at the same station. Similarly, we introduce *coupling edges*. Such edges connect depot events with platform events and indicate that the unit type will be coupled to other units to perform the outgoing subtrip. Coupling edges must respect a minimum coupling time, meaning a depot event cannot connect to the station platform event before the coupling time has elapsed.

Similarly, *decoupling edges* are introduced, each of which connects a platform event with a depot event. In contrast to coupling edges, the edges do not connect platform and depot events at the same station, thus eliminating the possibility of coupling a unit without taking a subtrip. Again, we assume a minimum duration on decoupling edges. This is modelled by connecting these edges to the target depot-station after the decoupling is completed. Note that the introduction of decoupling edges implicitly duplicates the number of subtrip edges that terminate at depot-stations. All coupling and decoupling edges are assigned an additional cost c_4 to reflect the price one must pay to perform a coupling. By counting the number of coupling edges used in any path from the source to the sink, one can determine how many times the unit is coupled and/or decoupled from a trip sequence. Finally, all coupling and decoupling edges must be adjusted by the dual contribution from Constraints (4.9) since any such edge indicates the consumption (or release) of depot track capacity. This dual contribution on any decoupling edge is given by $-l_u\check{\nu}_a$, with subtrip $a \in \mathcal{A}$ and $\check{\nu}_a = \sum_{a' \in \mathcal{A}_a^+} \nu_{a'}$. Here $\mathcal{A}_a^+$ gives the set of all subtrips in $\mathcal{A}$ arriving at depot $d = depot(a)$ whose arrival time is at least as large as that of subtrip a. Similarly, the depot track dual contribution on any coupling edge is $l_u\hat{\nu}_a$, where $\hat{\nu}_a = \sum_{a' \in \mathcal{A}_a^-} \nu_{a'}$. Here $\mathcal{A}_a^-$ is the set of all subtrips departing from d whose departure time is at least the arrival time of a.

Constructing such a network allows units to perform non-consecutive subtrips, include the coupling costs, and respect the minimum coupling durations. The last feature allows us to monitor when a unit type will be available to perform certain subtrips if (de)coupling decisions have been made. The shortest path in such this network identifies the column with minimum reduced cost.

Figure 4.7 shows how the network of Figure 4.6 can be extended to include depot subtrips as well as (de)coupling edges. Stations with depot tracks have depot event nodes and are coloured light gray. Note that all (de)coupling edges have not been included; however, an example of each can be seen in two of the three example paths provided. The solid black path depicts a unit which is coupled to the second outgoing subtrip at depot d_2. This unit performs four consecutive subtrips. The dashed path also illustrates a unit that performs four consecutive subtrips; however, this unit performs the first outgoing subtrip at station d_2. This unit is decoupled after its fourth subtrip. In contrast, the gray path gives an example of a unit that stays at depot-station d_1 during the planning period. As in Figure 4.6, a selection of arc weights have been included for expository purposes.

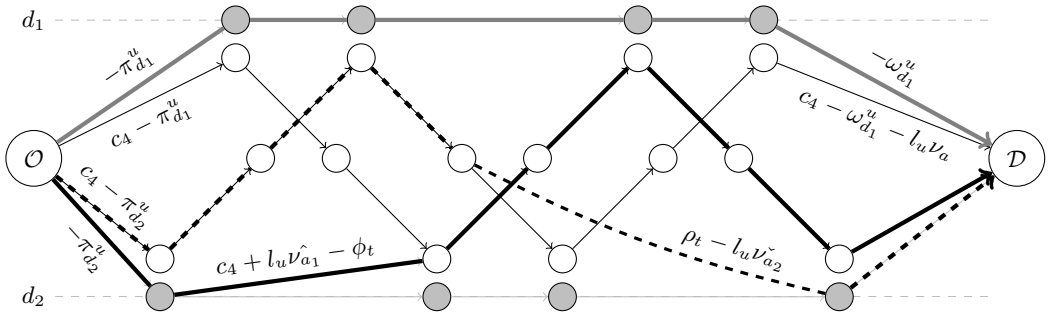


Figure 4.7: A simplified example of the acyclic time-space network showing some of the possible coupling arcs. Only two of the stations have an associated depot. Three possible unit paths have been highlighted.

Finally, including restrictions on unit mileage and coupling count we can consider maintenance restrictions. This may require a unit to have a path that does not exceed a certain maximum length, and control how many couplings – at most – we wish to permit any unit to perform. The nature of the time-space network in Figure 4.7 is such that we can also satisfy requests which ask specific units to end at specific depot-stations. Note, that adding potential deadheading trips corresponds to adding edges between station depots, and does not require additional variables or constraints in the master problem. As a side comment, the network will contain negative edge weights; however, negative cycles cannot not exist due to its acyclic nature. The shortest

path can therefore be found using a directed acyclic graph shortest path algorithm with linear time complexity. Furthermore, the network can, with some modification, be used to handle the situation described in Section 4.3.1.

4.4.3 Branching

The column generation process terminates with an optimal solution to the RRMP. If the variables satisfy the integral constraints, a feasible solution to the RSRP has been found, and we can update the upper bound and prune the current branch. This is, however, not always the case and we need to define branching rules in order to enforce integrality.

Branching on variables in the RRMP is not preferable in general, nor in our case. Forcing a fractional path variable to take the value 0 or 1 in separate branch nodes complicates the subproblem. In the 0 branch the subproblem will (without alterations) likely return the forbidden path as a new variable. Thus, in order to disallow specific paths we need to solve the *k-shortest (resource constrained) path problem*, just to make sure that we have the shortest path that is not forbidden by branching rules. In addition, the efficiency of branching on paths is uncertain as the total number of paths is extremely large. We propose branching on certain characteristics of the paths instead, which is easier to handle in the subproblems.

An interesting and effective branching rule has been proposed by Ryan and Foster [31] for the similar crew scheduling problem. Instead of branching on original or subproblem variables, which often is imbalanced or ineffective, a more balanced branching rule which can be handled entirely by the subproblem is proposed. In crew scheduling a split flow is found and a branching rule is set such that a column either covers two consecutive crew tasks or does not cover them both. Despite the similarities of the considered problem we cannot adopt this approach since our problem does not exhibit the required structure for this type of branching. In our case, we can enforce branches based on subtrip variables; but, the cover of a sub-trip is allowed to be greater than exactly one. Furthermore, applying the branching scheme on specific units introduces at least two disadvantages. Firstly, such a rule will not have a significant effect since we have many similar train units. Secondly, variations of the subproblem must be solved for every unit depending on its individual branching constraints.

We propose two sets of branching rules sufficient to ensure integrality. The first enforces integrality on the number of paths originating and terminating at a depot. The number of paths leaving a depot (as the first action) can never be fractional. For example, if the sum of units that depart from a station is fractional, say $f \in \mathbb{R}$, then we create two branches where the sum is $\leq \lfloor f \rfloor$ on the first branch and $\geq \lceil f \rceil$ in the other. The same holds for the number of paths entering (as the last action) a depot. This rule can be applied on two levels, either for a specific unit type or aggregated over all types. Our B&P framework imposes this rule on an aggregated level where possible, otherwise on a specific unit type.

Even if the paths respect the integral requirements above, the solution may still be fractional. The second set of branching rules enforces integrality on subtrip coverage. The subtrip cover is equal to the sum of paths that service a subtrip. We therefore ensure that the total subtrip cover, say $f \in \mathbb{R}$, is integer by imposing that the cover is $\leq \lfloor f \rfloor$ in one branch and $\geq \lceil f \rceil$ in the other. This branching rule can also be applied on two different levels. Where possible we aggregate over unit types, otherwise we create a branch based on a specific unit type.

We prioritize the branching candidates by always favouring the first set over the second, and by preferring aggregation over unit-specific rules. Note, however, when using strong branching, the prioritization is determined by the achieved bounds, see Section 4.5.

Theorem 4.1. *The proposed branching rules form a complete branching strategy given that the decoupling and coupling costs are strictly positive.*

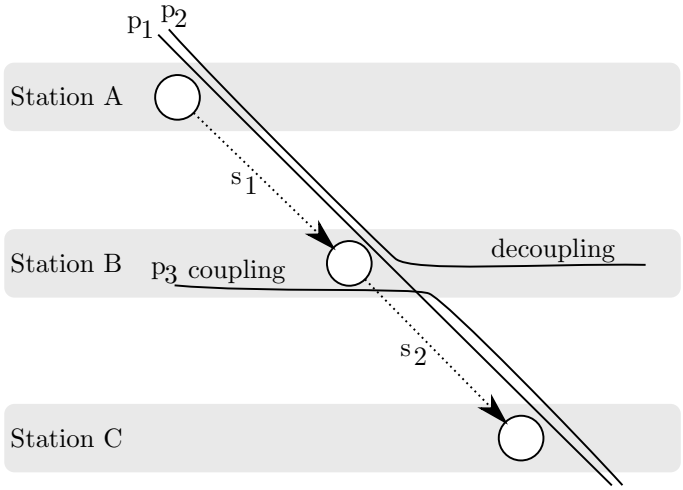


Figure 4.8: Illustration of a subset of a rolling stock schedule. Three fractional paths are shown covering two subtrips in the same trip sequence.

Assume we have an optimal solution that respects the constraints in RRMP and the branching rules. Further assume without loss of generality that no two paths are identical.

We prove, by contradiction, that every subtrip is covered by integral paths. If all subtrips are covered by integral paths, then if follows naturally that the paths flowing in and out of depot sources and target nodes are integral.

Assume that an optimal solution is found but one (or multiple) subtrip is covered by multiple fractional paths, say p_1 and p_2. These paths must diverge before or after some subtrip, otherwise they are identical. Assume the paths diverge after some subtrip, the other case is analogous. The case is illustrated in Figure 4.8.

After visiting a subtrip s_1 paths can either continue on the successor subtrip s_2 or decouple into the associate depot. Thus one of them will decouple. However, if a fraction of the flow decouples, then a different fractional path p_3 must be coupled on the successor path in order to maintain the integral sum of s_2. We note that the fractional path which is leaving s_1 or entering s_2 can consist of multiple paths, but the sum is fractional. We also note that, without loss of generality, paths p_1, p_2 or p_3 can further be a group of paths.

We now argue that this cannot be an optimal solution since a better alternative exists. Consider replacing p_2 and p_3 with new paths p_4 and p_5, where these paths are the results of swapping the subpaths of p_2 and p_3 after the (de)coupling position. Path p_4 now continues along p_1 on s_1 and s_2 while path p_3 simply stays in the depot instead of coupling on to s_2. The new resulting solution (p_1, p_4, p_5) omits a coupling and decoupling penalty, thus obtaining a lower cost. Thus we arrive at a contradiction, as the solution was assumed to be optimal.

■

4.5 Computational Results

In this section we present our computational results using the presented B&P framework. The solution method is benchmarked against data provided by DSB S-tog. We perform tests on both planned and disrupted instances. The essential differences between a normal and a disturbed case are the timetable and the objective. In the disruption instances we consider a disrupted timetable, but leave the objective structure unchanged. We note that a different objective, and possibly also

Name	Stops	Subtrips	Subtrips'	Weekday	Lines
Fri	28 719	3 259	1 086	Friday	A,B,Bx,C,E,F&H
Sat	20 474	1 795	706	Saturday	A,B,C&F
Sun	19 919	1 753	690	Sunday	A,B,C&F
Mon	28 017	3 196	1 074	Monday	A,B,Bx,C,E,F&H

Table 4.5: Four timetable and demand datasets provided by DSB S-tog. The columns respectively show the instance names, total number of stops, total number of natural subtrips, number of subtrips after preprocessing, Weekday and finally the lines that are running.

Scenario #1 - Leg Segment	
Contingency Plan	A.3.10 (Val - Fs)
Description	One track is blocked on one of the fingers
Cancelled lines	H
Turned lines	C1
Unchanged	A1, A2, B1, B2, Bx, C2, E1, E2

Table 4.6: Details of the leaf type disruption case. The disruption blocks one track between Valby and Frederikssund stations.

other extensions, might be more suitable when solving disruptions; however, this is considered out of scope for this paper.

4.5.1 Data

The test instances consist of real-life timetables and demands operated by DSB S-tog during the spring of 2014. DSB S-tog adopts a weekly periodic timetable. All weekdays follow the same timetables, while Saturday and Sunday have a different weekend timetable. On Friday and Saturday there are additional night-trains. In total we have four different cases: Monday-Thursday, Friday, Saturday and Sunday. Table 4.5 summarises the key characteristics of the instances.

Preprocessing has been performed to combine subtrips that arrive at or depart from stations without shunting options. Such subtrips can be merged since the composition cannot be changed. We note, that considering non-preprocessed instances can be misleading as the complexity of the instance is related to the number of *real* shunting decisions that can be made.

The number of possible disruption scenarios is very high as there can be multiple points of failure, disruption durations, start times and levels of severity. For the point of benchmarking our solution method we find it sufficient to benchmark two different types of disruptions of different severity. The topology of the infrastructure is star-shaped (see Figure 4.5), which means that two interesting cases appear: A disruption in the center section, or a disruption along one of the fingers. The disruption timetables are generated according to existing contingency plans. We test different start times and durations of both disruption types. Table 4.6 and 4.7 describe the two selected cases.

4.5.2 Lower Bound

The gap between the optimal solution and the root LP relaxation is in some cases relatively high - even above 10%. This is not surprising as the problem has many similarities with the Integer Multicommodity Flow Problem (MCFP) which has a similar property, see [4].

Scenario #2 - Central Segment	
Contingency Plan	A.0.16 (Kh - Kk)
Description	One track is blocked in the central segment
Cancelled lines	A2, B2, Bx, C1, E2, H
Turned lines	A1, B1, C2, A1, C1
Unchanged	E1

Table 4.7: Details of the central type disruption case. The disruption blocks one track between København H and Østerport stations

The bound can be strengthened by solving a MIP formulation of the relaxed problem. For generating better bounds we solve a MIP that essentially is a relaxed version of the formulation presented by Fioole et al. [19]. The MIP model assigns one composition to every trip, using the rolling stock fleet. The output is an anonymous flow of the available units, where no restriction is set on composition changes. We further improve this bound by including depot capacities, i.e. Constraints (4.9) in an appropriate and modified form.

The correctness of this bound is apparent as the flow model solves the same problem, only not accounting for the structure of the generated paths. A valid solution to the path formulation is feasible in the flow model.

Our preliminary results show that this lower bound is stronger than the LP relaxation of the master problem. The model is solved in a few seconds or up to one minute. We therefore adopt this bound in this section.

4.5.3 Tuning

Every B&P framework has many tuning-options. Without going into details, we briefly mention some initial tuning experiments which have been performed.

The columns of the initial master problem are chosen carefully to ensure feasibility. We tried inserting different heuristic columns that cover sequences of subtrips. Inserting such heuristic columns did not, however, seem to improve convergence nor turn out to be beneficial in any way.

An experiment was performed inserting the lower bound as a constraint on the objective in the master problem. This experiment also did not show any consistent improvement on data sets.

Preliminary tests revealed that the branching rules are not very effective. The number of nodes in the B&B tree is high. We have four different types of branching rules and the number of possible branches in every fractional node is large. We implemented and tested *Strong Branching* which proved to efficiently reduce the B&B tree in general. We adopt a variant of *Full Strong Branching* where all possible branching candidates are generated and ordered by fractional distance [2]. The candidates are evaluated in order until no improvement has been made for a certain number of iterations.

Generating multiple columns in every iteration of the B&P framework improves convergence, compared to only inserting one path with the lowest reduced cost. However, generating too many columns proved to have a negative impact on runtime. For every depot we find and use the lowest reduced cost path per unit type.

Depending on the considered objective the instance root relaxation is, more often than not, far from integrality. In order to get an fast initial solution we tried to add a step which solves the

Instance	Horizon	Subtrips	Columns	Time
Fri	4:40-27:10	1 086	1 246	6
Sat	2:00-27:10	706	411	1
Sun	2:00-25:38	690	350	1
Mon	4:37-25:38	1 074	1 584	7

Table 4.8: Optimal solutions obtained only considering cancellation costs and a small cost for couplings. The elapsed time is measured in seconds.

root node relaxation as a MIP. In many cases this proved to find very good initial solutions in a few seconds. We therefore adopt this approach.

A B&P framework is ideal for parallel execution. The results, however, showed that the performance can sometimes decrease, especially if small instances are considered. Our conclusion is that parallel execution is most beneficial when the instances are large, and when the B&B tree becomes large. For the sake of clarity, the following benchmarks are run using a single thread.

A comparison between using CPLEX and the COIN-OR simplex solver (CLP) for solving the master relaxation showed that the latter is generally faster and more robust. We observed a faster and steadier convergence using CLP. However, we also observed that CPLEX in a few cases was able to more quickly find an incumbent. In our experiments we use CLP for solving the master relaxations.

4.5.4 Planning Benchmark

The RSRP includes multiple objectives; however, we consider a single objective function which in turn is a weighted sum of all objectives. Finding the correct or appropriate weights for each objective is far from trivial.

The most important objective is the total number of trip cancellations; all other objectives are of small importance in comparison. We only expect to see cancellation during disruptions. Next, we consider end-of-day balance deviations to be second most important since rebalancing rolling stock incurs a large economical cost in terms of equipment and crew. Finally, a good balance between demand cover, operational cost and the number of (de)couplings is sought.

In the first experiment we benchmark the performance of the framework solely considering the number of cancellations. An insignificantly small penalty is, however, set on the number of couplings since preliminary experiments show that the solution method performs worse without. Without a coupling cost the number of alternative optimal paths in the subproblem increases drastically; the number of different paths (with the same cost) negatively affects the integrality of the master problem. The results for the first benchmark are shown in Table 4.8. The results show that we are able to solve the planning instances to optimality within a few seconds. In all cases there is a gap of 0% between the relaxed root node and optimal value. This suggests that our formulation has some good properties using the considered objectives. All instances were solved to optimality before branching could occur. This experiment provides evidence that the model performs well when the sole goal is to find a certain (integral) cover, e.g, if all subtrip compositions are given beforehand.

A minimal seat-shortage is desirable and experiments performed using a small coupling cost and different shortage costs are shown in Table 4.9. The results show that the algorithm has difficulty solving the problem to optimality. Seat cover is at 100% which means seat shortage contributes with a zero cost to the objective. A 5% termination gap was used since the framework has a hard time improving this gap, even given a time limit of 1 hour. The gap is therefore solely caused by the number of couplings performed. The relatively large root gap is not unexpected

Instance	Subtrips	Time	Columns	Gap	Root Gap	Cover
Fri	1 086	31	3 328	4.0%	13.6%	100.0%
Sat	706	1	514	0.0%	0.0%	100.0%
Sun	690	2	472	0.0%	2.8%	100.0%
Mon	1 074	19	2 678	1.5%	9.1%	100.0%

Table 4.9: Solutions obtained only considering cancellation costs, a small shunting cost and a seat shortage cost. The elapsed time is measured in seconds. *Gap* and The *Root Gap* show respectively the relative optimality bound and the relative distance to the root node relaxation. *Cover* shows the percentage of covered seats in the found solution.

Instance/Penalty	0.0	0.05	0.1	1.0
Fri	87.04%	99.94%	99.95%	100.00%
Sat	96.90%	100.00%	100.00%	100.00%
Sun	98.16%	100.00%	100.00%	100.00%
Mon	83.14%	99.95%	100.00%	100.00%

Table 4.10: Seat demand coverage obtained using increasing seat shortage penalties.

since fractional unit paths are able to cover seat demand more cost-efficiently than integral paths which have to pay full coupling penalties. In a planning context, the number of couplings is not a primary concern compared to scheduling new couplings during the day of operation.

In planning the goal is to meet the expected demand. A seat-coverage of 100% is ideal; however, if the cost of covering 99% is significantly less, then this is also an acceptable solution. Table 4.10 shows the seat cover using varying seat shortage costs. As expected, the seat cover increases as the cost increases.

Minimizing cancellations, shunting operations and seat shortages can lead to expensive schedules since train services can be executed with too much capacity. Minimizing the mileage is therefore also essential. In the final planning benchmark we use varying mileage costs. The results are shown in Table 4.11. Note, that *mileage cover* is relative to the travel-distance of assigning one unit to all subtrips (including dead-heading). Thus 100% is the minimum if no cancellations are made. Using different penalties the total mileage decreases as the mileage penalty increases. As we aim for a high seat cover the trade-off is between the number of couplings and mileage. The total runtime ranges up to a few minutes. We observe that the (root) gap improves as the (de)coupling penalty diminishes (relative to mileage cost), and more columns are necessary to prove (LP) optimality. This is due to the fact that mileage and (de)coupling become a more equally priced in the objective.

The planned cover and mileage at DSB S-tog is consistent with the our results and only differ slightly, see Figure 4.11. We note that these plans are achieved using optimization tools; thus we cannot expect to find plans that are significantly better. In general, the planners have achieved a high seat cover and also a relatively low mileage cover at the cost of more (de)couplings.

4.5.5 Disruption Benchmark

Adopting the parameter settings found in the planning benchmarks, we study a number of disruption cases. In this setting we consider short runtimes essential as this is a crucial precondition when resolving a disruption.

We benchmark two types of disruptions (and corresponding contingency plans) as mentioned in Section 4.5.1. We consider disruption durations of between 1 and 4 hours with three different

Instance	Penalty	Time	Columns	Gap	Root	Seat	Mileage	Couplings
Fri	0.01	100	6 068	2.9%	12.4%	100.0%	126.5%	282
Fri	0.10	113	6 286	1.5%	10.1%	100.0%	126.2%	278
Fri	1.00	590	14 158	0.9%	6.9%	100.0%	119.3%	294
Sat	0.01	5	1 295	0.0%	0.0%	100.0%	100.6%	116
Sat	0.10	5	1 317	0.0%	0.0%	100.0%	100.6%	116
Sat	1.00	20	3 860	0.0%	0.0%	100.0%	100.6%	116
Sun	0.01	4	1 180	0.0%	2.8%	100.0%	100.8%	120
Sun	0.10	4	1 316	0.0%	2.4%	100.0%	100.8%	120
Sun	1.00	23	3 842	0.0%	1.2%	100.0%	100.8%	120
Mon	0.01	45	4 850	1.4%	9.1%	100.0%	126.0%	274
Mon	0.10	48	5 126	0.7%	8.2%	100.0%	125.8%	272
Mon	1.00	255	10 695	0.7%	7.3%	100.0%	119.1%	296

Table 4.11: Shows the results obtained for the instances using increasing mileage cost penalties. A termination gap of 5% was set. The column show the instance, mileage penalty, runtime, number of columns generated, optimality gap, root lp relaxation optimality gap, seat coverage, mileage cover and number of (de)couplings.

	Planned		Realized		
Instance	Unit Cover	Demand cover	Unit Cover	Demand cover	Couplings
Fri	117.44%	99.91%	117.75%	99.32%	371
Sat	112.41%	99.96%	112.41%	99.81%	205
Sun	109.54%	99.88%	109.71%	99.69%	186
Mon	116.08%	99.99%	116.09%	91.71%	459

Table 4.12: Shows the statistics of the planned and realized solution for the datasets. The number of couplings are unavailable in the planned solution and estimated based on the routes in the realized solution.

					Quality				
	Disruption	#	Time	Cols	Gap	Root	Cover	Seat	Mileage
Fri	9:00-10:00	823	5	2 563	0.9%	3.4%	100.0%	99.0%	122.9%
Fri	9:00-13:00	814	6	2 608	0.0%	3.2%	100.0%	99.4%	124.8%
Fri	11:00-12:00	703	5	2 234	0.3%	3.8%	100.0%	99.1%	121.3%
Fri	11:00-15:00	692	5	2 483	0.3%	2.1%	100.0%	99.0%	122.7%
Fri	15:00-16:00	456	3	1 456	0.0%	0.8%	100.0%	98.9%	118.3%
Fri	15:00-19:00	442	3	1 362	0.0%	1.2%	100.0%	99.2%	118.8%
Mon	9:00-10:00	807	5	2 642	0.1%	2.6%	100.0%	98.9%	126.1%
Mon	9:00-13:00	798	6	3 391	0.6%	2.8%	100.0%	99.1%	126.3%
Mon	11:00-12:00	687	6	2 460	0.9%	4.0%	100.0%	98.8%	124.5%
Mon	11:00-15:00	676	5	2 854	0.5%	2.9%	100.0%	98.9%	121.6%
Mon	15:00-16:00	440	5	1 626	0.0%	2.4%	100.0%	98.2%	119.0%
Mon	15:00-19:00	426	3	1 416	0.7%	2.1%	100.0%	98.2%	121.1%

Table 4.13: Results of solving the first type of disruption case. The first columns show the instance, disruption duration, number of subtrips and runtime. *Cols* shows the number of generated columns. The next columns show the optimality gap w.r.t. the lower bound and root node relaxation. Finally, the last column measure the quality of the solution w.r.t. covered subtrips, covered seats and train unit mileage.

starting points: 9:00, 11:00 and 15:00. The selected periods overlap with the peak periods in different ways. The weekend instances *Sat* and *Sun* are excluded from consideration as they are smaller and arguably easier to solve.

Given the type of disruption, duration and starting point, we generate an updated (disrupted) timetable using existing DSB S-tog contingency plans. The planned solution is assumed to have been executed up until the starting point, and the remainder of the day must be solved. As argued previously, the uncertainty of the disruption can be handled by adopting a rolling horizon scheme, i.e., every time new information is available a new updated timetable is made.

The results of the benchmarks are shown in Tables 4.13 and 4.14. The framework was set to terminate and report the best found solution within 5 minutes or when a optimality gap of 1% is proven. Note, that since we solve the remainder of the day this means that longer durations and later starting points result in smaller instances. Note that (de)couplings are given a higher penalty during the disruptions, thus favouring higher robustness over seat and mileage cover.

In Table 4.13 the results of activating the contingency plan *A311* with multiple duration and starting points can be seen. We obtain very good solutions within a few seconds. Due to a decrease in the number of subtrips, a low runtime is expected. Again, the lower bound shows a noticeable improvement compared to the relaxation of the root node. In all cases the final solutions are found after solving the root node as a MIP. A full cover is obtained in all cases, which means that all trips in the disrupted timetable are possible to cover. The cover metric does not include the trips cancelled due to switching to the contingency plan. The mileage used and seat coverage is close to what we observed during planning.

Table 4.14 shows the results of activating the contingency plan *A016* with multiple duration and starting points. Again, good solutions are found within a few seconds. Again, in all cases the final solutions are found after solving the root node as a MIP. Now the strength of the flow bound is more apparent than before, in the worst case the improvement is up to approximately 15% compared to the relaxation of the root node. This change is probably correlated to reduced trip cover. We observe that some trips are now cancelled in the disrupted timetable, and a fractional solution found in the root node relaxation is able to cover more trips, seats and use less couplings and mileage. The mileage usage is still comparable to our previous results. The slightly lower seat cover is partly due to the necessity of cancelling trans but also due to a more turbulent fleet position.

					Quality			
	Disruption	#	Time	Cols	Gap	Root	Cover	Seat	Mileage
Fri	9:00-10:00	836	11	4 096	0.3%	15.1%	99.8%	98.9%	122.4%
Fri	9:00-13:00	782	18	6 068	0.5%	14.7%	99.8%	98.7%	119.4%
Fri	11:00-12:00	713	8	3 051	0.6%	16.7%	99.8%	98.9%	122.1%
Fri	11:00-15:00	657	10	3 939	0.2%	14.8%	99.7%	98.4%	117.5%
Fri	15:00-16:00	468	6	2 379	0.3%	2.1%	99.6%	97.2%	113.8%
Fri	15:00-19:00	402	5	1 727	0.0%	1.9%	99.8%	98.4%	115.3%
Mon	9:00-10:00	820	13	4 129	0.4%	2.7%	99.8%	98.4%	121.6%
Mon	9:00-13:00	766	14	5 827	0.1%	2.5%	99.8%	98.8%	118.3%
Mon	11:00-12:00	697	10	3 665	0.5%	16.5%	99.8%	98.8%	122.4%
Mon	11:00-15:00	641	8	4 066	0.2%	14.3%	99.7%	98.4%	118.6%
Mon	15:00-16:00	451	6	2 239	0.0%	1.5%	99.6%	97.5%	119.1%
Mon	15:00-19:00	385	6	2 053	0.2%	1.9%	99.8%	97.2%	118.2%

Table 4.14: Results of solving the second type of disruption case. The first columns show the instance, disruption duration, no of subtrips and runtime. *Cols* shows the no of generated columns. The next columns show the optimality gap w.r.t. the lower bound and root node relaxation. Finally, the last column measure the quality of the solution w.r.t. covered subtrips, covered seats and train unit mileage.

4.5.6 Maintenance Constraints

Different unit specific constraints can be enforced naturally in our framework. To demonstrate, we give some results of adding mileage restrictions to rolling stock units. In the experiments one type of maintenance requirement at DSB S-tog, where all trains must undergo a *sanding* inspection every ten megameters, is considered. This inspection concerns the the wheel slide protection system.

We benchmark a set of generated instances as no real-life data is available. A single rolling stock unit usually drives up to one megameter during a normal weekday and, on average, half a megameter. Ideally, every unit is assigned a random mileage-before-maintenance (MBM) between zero and ten mega-meters; however, for practical reasons we assign a random MBM between zero and one megameters on a selected subset of the fleet using an appropriate random probability. Note, there is no reason to put restrictions on units that never will reach the limit. Using different seeds (for the random number generator) we generated the different maintenance instance based on the *Mon* planned instance with a mileage penalty of 1.0, c.f. Table 4.11.

The results of the benchmark are shown in Figure 4.15. Note that it is not possible to provide an exact comparison as we only find a heuristic solution with a proven low gap. With a noticeable increase in runtime, the demands of all instances were covered fully, and the total mileage is comparable to the original instance. The number of (de)couplings is similar, if not slightly higher. In addition some variation comes with the random selection of maintenance units.

The optimal LP-relaxed solution is found at the root node, and can be used as an indication of the weakened fleet. Here, compared to the original instance we did not notice a significant increase in the relaxed objective value (i.e. overall cost) with more mileage restricted units. We conclude that the fleet is able to absorb the added restriction without incurring high penalties.

4.6 Conclusions

In this paper we presented and investigated a Branch-and-Price framework for optimizing the Rolling Stock Rescheduling Problem, and benchmarked both planned and disrupted real-life DSB S-tog instances.

Probability	Time	Columns	Gap	Couplings	Seat	Mileage
5%	464	15 135	1.4%	304	100.0%	118.0%
5%	461	15 135	1.4%	304	100.0%	118.0%
5%	450	17 227	2.0%	308	100.0%	118.1%
5%	351	14 441	1.5%	304	100.0%	118.3%
5%	273	11 832	1.9%	306	100.0%	117.9%
10%	401	18 982	1.6%	298	100.0%	120.3%
10%	400	18 982	1.6%	298	100.0%	120.3%
10%	413	16 196	2.6%	312	100.0%	118.2%
10%	621	20 579	1.6%	302	100.0%	119.1%
10%	526	20 358	2.1%	310	100.0%	117.7%
15%	690	25 389	1.5%	302	100.0%	118.8%
15%	686	25 389	1.5%	302	100.0%	118.8%
15%	490	18 522	2.0%	308	100.0%	118.1%
15%	346	16 392	2.6%	310	100.0%	118.7%
15%	222	14 451	2.2%	308	100.0%	118.5%

Table 4.15: Results of maintenance cases. The first column shows the probability of a rolling stock units to receive a limiting mileage during the planning horizon. The other columns respectively show runtime, number of generated columns, optimality gap, number of (de)couplings, seat cover and mileage cover.

The RSRP is a multi-criteria optimization problem that is usually solved using a linear weighted objective. However, we observed that the objective must be carefully tuned in order to achieve the desired balance between operational cost, shunting and seat shortages. In this paper we found a balance that achieves a demand cover very close to 100%, while minimizing operational costs.

The B&P framework is able to solve the instances to optimality, but this proves to be time-consuming in certain settings. However as discussed, proven optimality is not essential since finding the correct objective weight balance is non-trivial. The benchmark revealed that we can find solution with a small proven optimality gap within few minutes for the hardest planning instances, and within 20 seconds for the disruption instances. In general, the test shows that the framework is able to find good solutions very fast, but on the hand not able to close the optimality gap efficiently. A MIP based lower bound efficiently improved the optimality gap.

Future research we will focus on several things. One obvious extension is to impose train composition and shunting restrictions. This would allow the train operators to more explicitly control how already running train compositions can be upgraded or reduced in order to improve robustness or feasibility. The proposed mathematical model mainly consists of several soft constraints with non-integral right hand sides. Enforcing hard constraints would limit the solution space and replacing the demand constraints with the most desired train composition would arguably improve the solution method. Finally, it would be interesting to extend the current models in order to model already used maneuvers at DSB S-tog during disruptions. Examples include allowing trains to turn earlier than planned, and allowing two trains to swap services when meeting at the same station.

Bibliography

[1] E. Abbink, B. van den Berg, L. Kroon, and M. Salomon. Allocation of railway rolling stock for passenger trains. *Transportation Science*, 38:33–41, 2004.

[2] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Reserch Letters*, 33(1):42–54, 2005.

[3] Arianna Alfieri, Rutger Groot, Leo Kroon, and Alexander Schrijver. Efficient circulation of railway rolling stock. *Transportation Science*, 40(3):378–391, 2006.

[4] Barnhart, Hane, and Vance. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Oper. Res. (USA)*, 48(2):318–326, 2000.

[5] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[6] Nejib Ben-Khedher, Josephine Kintanar, Cecile Queille, and William Stripling. Schedule optimization at sncf: from conception to day of departure. *Interfaces*, 28(1):6–23, 1998.

[7] Ralf Borndörfer, Markus Reuther, Thomas Schlechte, and Steffen Weider. A Hypergraph Model for Railway Vehicle Rotation Planning. In Alberto Caprara and Spyros Kontogiannis, editors, *11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 20 of *OpenAccess Series in Informatics (OASIcs)*, pages 146–155, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[8] Gabriella Budai, Gábor Maróti, Rommert Dekker, Dennis Huisman, and Leo Kroon. Rescheduling in passenger railways: the rolling stock rebalancing problem. *Journal of Scheduling*, 13:281–297, 2010.

[9] Valentina Cacchiani, Alberto Caprara, Laura Galli, Paolo Toth, Leo Kroon, Gábor Maróti, and Leo Kroon. Railway rolling stock planning: Robustness against large disruptions. *Transportation Science*, 46(2):217–232, 2012.

[10] Valentina Cacchiani, Alberto Caprara, and Paolo Toth. Solving a real-world train-unit assignment problem. *Math. Program.*, 124(1-2):207–231, 2010.

[11] Luis Cadarso and Ángel Marín. Robust routing of rapid transit rolling stock. *Public Transport*, 2(1-2):51–68, 2010.

[12] Luis Cadarso and Ángel Marín. Robust rolling stock in rapid transit networks. *Computers & Operations Research*, 38(8):1131–1142, 2011.

[13] Luis Cadarso, Ángel Marín, and Gábor Maróti. Recovery of disruptions in rapid transit networks. *Transportation research. Part E, Logistics and transportation review*, 53:15–33, 2013.

[14] A. Caprara, L.G. Kroon, M. Monaci, M. Peeters, and P. Toth. Passenger railway optimization. In C. Barnhart and G. Laporte, editors, *Handbooks in Operations Research and Management Science*, volume 14, chapter 3, pages 129–187. Elsevier, 2007.

[15] Jean-François Cordeau, François Soumis, and Jacques Desrosiers. A benders decomposition approach for the locomotive and car assignment problem. *Transportation Science*, 34(2):133–149, May 2000.

[16] Jean-François Cordeau, François Soumis, and Jacques Desrosiers. Simultaneous assignment of locomotives and cars to passenger trains. *Oper. Res.*, 49(4):531–548, July 2001.

[17] Jean-François Cordeau, Guy Desaulniers, Norbert Lingaya, François Soumis, and Jacques Desrosiers. Simultaneous locomotive and car assignment at {VIA} rail canada. *Transportation Research Part B: Methodological*, 35(8):767 – 787, 2001.

[18] Jacques Desrosiers, Yvan Dumas, Marius M. Solomon, and François Soumis. Chapter 2 time constrained routing and scheduling. In C.L. Monma M.O. Ball, T.L. Magnanti and G.L. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, pages 35 – 139. Elsevier, 1995.

[19] Pieter-Jan Fioole, Leo Kroon, Gábor Maróti, and Alexander Schrijver. A rolling stock circulation model for combining and splitting of passenger trains. *European Journal of Operational Research*, 174(2):1281–1297, 2006.

[20] Julie Groth, Daniel Potthoff, Jens Clausen, Dennis Huisman, Leo Kroon, Gábor Maróti, and Morten Nielsen. Disruption management in passenger railway transportation. In Ravindra Ahuja, Rolf Möhring, and Christos Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 399–421. Springer Berlin / Heidelberg, 2009.

[21] Julie Jespersen Groth and Jens Clausen. *Optimal Reinsertion of Cancelled Train Lines*. 2006.

[22] Julie Jespersen Groth, Daniel Potthoff, Jens Clausen, Dennis Huisman, Leo Kroon, Gábor Maróti, and Morten Nyhave Nielsen. Disruption management in passenger railway transportation. *Robust and Online Large-Scale Optimization*, pages 399–421, 2009.

[23] Stefan Irnich. Resource extension functions: properties, inversion, and generalization to segments. *OR Spectrum*, 30(1):113–148, 2008.

[24] Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In Guy Desaulniers, Jacques Desrosiers, and MariusM. Solomon, editors, *Column Generation*, pages 33–65. Springer US, 2005.

[25] Leo Kroon and Dennis Huisman. Algorithmic support for railway disruption management. In Jo A.E.E. Nunen, Paul Huijbregts, and Piet Rietveld, editors, *Transitions Towards Sustainable Mobility*, pages 193–210. Springer Berlin Heidelberg, 2011.

[26] N. Lingaya, JF Cordeau, G. Desaulniers, J. Desrosiers, and F. Soumis. Operational car assignment at via rail canada. *Transportation Research Part B: Methodological*, 36(9):755 – 778, 2002.

[27] Lars Kjær Nielsen, Leo Kroon, and Gábor Maróti. A rolling horizon approach for disruption management of railway rolling stock. *European Journal of Operational Research*, 220(2):496–509, 2012.

[28] Marc Peeters and Leo Kroon. Circulation of railway rolling stock: a branch-and-price approach. *Computers & operations research*, 35(2):538–556, 2008.

[29] Daniel Potthoff, Dennis Huisman, Daniel Potthoff, Dennis Huisman, and Guy Desaulniers. Column generation with dynamic duty selection for railway crew rescheduling. *Transp Sci*, 44(4):493–505, 2010.

[30] Natalia J. Rezanova and David M. Ryan. The train driver recovery problem—a set partitioning based model and solution method. *Computers & Operations Research*, 37(5):845 – 856, 2010.

[31] David M. Ryan and Brian A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*, pages 269–280. North-Holland, 1981.

[32] Alexander Schrijver. Minimum circulation of railway stock. *CWI Quarterly*, 6:205–217, 1993.

[33] Koorush Ziarati, François Soumis, Jacques Desrosiers, Sylvie Gélinas, and André Saintonge. Locomotive assignment with heterogeneous consists at {CN} north america. *European Journal of Operational Research*, 97(2):281 – 292, 1997.

Chapter 5

A Comparison of Two Exact Methods for Passenger Railway Rolling Stock (Re)Scheduling

Jørgen T. Haahr* Joris C. Wagenaar† Lucas P. Veelenturf‡ Leo G. Kroon†

*Department of Management Engineering, Technical University of Denmark,
Produktionstorvet, Building 426, DK-2800 Kgs. Lyngby, Denmark
jhaa@dtu.dk

†Rotterdam School of Management, Erasmus University,
PO Box 1738, 3000 DR Rotterdam, The Netherlands
jwagenaar@rsm.nl, lkroon@rsm.nl

‡School of Industrial Engineering, Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
l.p.veelenturf@tue.nl

[1] **Abstract** The assignment of rolling stock units to timetable services in passenger railways is an important optimization problem that has been addressed by many papers in different forms. Solution approaches have been proposed for different planning phases: strategic, tactical, and operational planning. In this paper we compare two approaches within two operational planning phases (i.e. the daily and the real time planning). The first exact approach is based on a Mixed Integer Linear Program (MILP) which is solved using CPLEX. The second approach is an extension of a recently introduced column generation approach. In this paper, we benchmark the performance of the methods on networks of two countries (Denmark and The Netherlands). We use the approaches to make daily schedules and we test their real time applicability by performing tests with different disruption scenarios. The computational experiments demonstrate that both models can be used on both networks and are able to find optimal rolling stock circulations in the different planning phases. Furthermore, the results show that both approaches are sufficiently fast to be used in a real-time setting.

[1]Submitted to *Transportation Research Part E*, October 2015

5.1 Introduction

In passenger railway optimization one of the main goals is to make efficient plans that can accommodate all passengers or, if that is not possible, that minimize the seat shortages. In the planning process, a railway operator tries to match the demand by first selecting an appropriate timetable followed by a matching rolling stock schedule. We focus in this research on the second stage where a rolling stock schedule must be found given the timetable and passenger demand. Not only the planning process, in which a long time ahead a rolling stock schedule must be determined, is considered, but also the real time construction of rolling stock schedules due to disruptions.

If during operations an unexpected event causes the timetable, the rolling stock, and crew schedule to become infeasible, then these schedules need to be rescheduled. To ensure that the operations are not deadlocked, new feasible schedules must be found promptly. For a general overview of models for railway rescheduling during disruptions and disturbances we refer the reader to Cacchiani et al. [8], and for an overview on disruption management processes in general we refer to Groth et al. [17]. The major difference between rolling stock scheduling in the planning phase and rolling stock rescheduling during disruptions is the time available to come up with a solution. Next to that, during the rescheduling phase there is less flexibility since the trains are already running, and choices made before the disruption occurred cannot be reversed. A natural consequence is that it may not be possible to assign rolling stock to all train services, i.e., some train services may need to be cancelled. Therefore in the rescheduling case it is highly undesirable, but considered feasible, to assign no rolling stock to some services.

There is a body of literature tackling the problem of assigning rolling stock to passenger train services. These papers focus either on the planning process or on the rescheduling process. The majority of the approaches existing in literature considers one specific (and national) network, and does not benchmark against other approaches. In this paper we want to make a start with comparing different approaches, as well as comparing them on different networks. We hope to encourage other researchers to do the same.

Two main categories of rolling stock problems have been studied for passenger railways in literature. The first consists of assigning both carriages and locomotives to trips. Each carriage can be coupled individually and independently to a convoy (or composition), but at least one locomotive is required to pull the convoy. The other branch of research consists of assigning self-propelled train units that need not be pulled by a locomotive. These units consist of a fixed number of carriages and have their own traction engines. It is common that these train units can be coupled together to form larger train compositions.

The problem of assigning locomotives and/or carriages to trains can be applied to passenger trains (see e.g. [6], [12], [11], and [19]), but also to freight trains (see e.g. [1], [5], [15], [23], and [24]). The problem of (re)scheduling self-propelled units is also considered in multiple publications (see e.g. [3], [4], [7], [9], [10], [14], [16], [20], and [21]). For more references, with respect to papers considering rolling stock (re)scheduling approaches, we refer the interested reader to Cacchiani et al. [8] and Piu and Speranza [22].

In this paper we consider two rolling stock (re)scheduling approaches for self-propelled train units. The first approach is based on the algorithm introduced by Fioole et al. [14]. This approach makes use of a general purpose solver (CPLEX) to solve a Mixed Integer Linear Program (MILP) for rolling stock (re)scheduling.

The second approach is an extension of the algorithm introduced by Haahr et al. [16] which involves path generation for individual rolling stock units. This algorithm makes use of column generation to solve a MILP. We note that Cacchiani et al. [7] also investigated a path based model for the rolling stock scheduling problem. Column generation in combination with rolling stock (re)scheduling is also consider by Peters and Kroon [21]. However, a different decomposition method was performed that did not involve the generation of paths for individual rolling stock

units.

Like, in Cacchiani et al. [7], the method of Haahr et al. [16] does not consider the order of rolling stock units within the train compositions. However, the order is of significant importance to determine which units can be decoupled from a composition. For example, at certain stations only the unit at the rear of a composition can be decoupled. Therefore we introduce a new formulation which considers the order of rolling stock units within compositions. This path based formulation requires adaptations to the column generation heuristic introduced by Haahr et al. [16]. To improve the performance in terms of computation time, a row generation variant of the algorithm is introduced as well.

The advantage of the path based formulation based on Haahr et al. [16] is that it models rolling stock unit duties (a list of subsequent trips) explicitly, thereby enabling dealing with unit specific constraints naturally; a good example of such constraints are units requiring a maintenance appointments at workshops. An optimal solution of the MILP formulation of Fioole et al. [14] only contains information on which compositions are assigned to which trips and how compositions change between trips. However, it does not produce paths for each individual rolling stock unit. In a post processing step, a simple heuristic can construct these individual routes, because an integer flow can always be decomposed into unit valued path flows (see Ahuja et al. [2]). However, taking constraints on individual units into account is not possible. For a discussion and other solutions for this problem we refer the reader to Wagenaar et al. [25].

Although the Mixed Integer Linear Programs of the two approaches are different, they are set up such that they solve exactly the same problem. Since both solution methods are exact solution methods, they will come to solutions with the same objective value. Note that the solutions can differ if multiple solutions with the same objective value exist.

We benchmark both approaches on rolling stock scheduling and rescheduling instances of Netherlands Railways (NS) and the Copenhagen Suburban Railway Operator DSB S-tog. In the scheduling instances a timetable and the passenger demand are given. The methods must assign rolling stock compositions to every train service such that the seat coverage of passenger demand is satisfactory while minimizing operational costs.

For the rescheduling instances, the original rolling stock schedule, an updated timetable, and the passenger demand are given. In this paper we assume that the passenger demand will not change due to a disruption - modelling passenger demand during a disruption is out of the scope of this paper. In these instances the main objective is to assign an appropriate rolling stock composition to as many train services as possible. However, it is no longer a hard constraint that all train services need a rolling stock composition assigned to them. The secondary objectives are to cover the passenger demand as well as possible and to deviate as little as possible from the original schedule. All deviations from the original schedule require additional shunting movements like couplings and uncouplings of train units. Unplanned or cancelled shunting movements require additional communication and coordination with shunting personnel; in some cases even additional shunting personnel must be arranged. Introducing deviations to the planned schedule is thus not preferred, especially since the available time is limited.

The remainder of this paper has the following structure. First, the contributions of this paper are discussed in Section 5.1.1. The problem description and assumptions are discussed in Section 5.2. The mathematical formulations for the two solution approaches are discussed in Section 5.3. In Section 5.4 computational experiments are presented. The paper is concluded in Section 5.5.

5.1.1 Contribution

This paper has several methodological and practical contributions. The methodological contributions are *i)* the extension of: an existing column generation formulation, which is bench-

marked against a compact formulation, and *ii)* the introduction of a new row generation approach. The new mathematical formulation is a path based MILP formulation for the rolling stock (re)scheduling problem. In this formulation the order of units within compositions is taken into account. The column generation approach is an extended version of the approach of Haahr et al. [16]. The extension was necessary to incorporate the order of units within compositions. Finally, a row generation method is adopted for a significant speedup in runtime of the solution approach.

The practical contributions are: *i)* realistic tests on DSB S-tog and Netherlands Railways instances, *ii)* a comparison between different rolling stock (re)scheduling approaches, and *iii)* comparisons of different instances within different countries. Furthermore, we encourage other railway operations researchers to perform benchmarks as well. This is the first work that actually incorporates the order of units within compositions in test instances of DSB S-tog. The transition (i.e. composition) rules have been made in co-operation with DSB S-tog. A comparison is made between different rolling stock (re)scheduling approaches by testing them on the same data sets. These approaches are benchmarked on two different railway networks with cyclic timetables, namely a large train service network in the Netherlands and the suburban network in Copenhagen, Denmark.

5.2 Problem description

In this section we describe the rolling stock (re)scheduling problem in more detail, and the assumptions we make.

The rolling stock *scheduling* problem consists of assigning a rolling stock composition to every trip in the timetable of one planning horizon, e.g., one day of operation. The rolling stock *rescheduling* problem consists of assigning rolling stock compositions to as many trips as possible for the remainder of the day in case of a disruption. A trip is a part of a train service, as specified by the timetable, between two major stations where the composition of the train can be changed. A trip consists of a departure station, departure time, arrival station, and arrival time. Furthermore, a composition is an ordered set of coupled rolling stock units. The assignment of rolling stock compositions should be done such that it minimizes the number of seat shortages for passengers, the total number of carriage kilometers, and several other objectives. Consequently, the rolling stock (re)scheduling problem essentially represents a multi-criteria decision problem.

For each trip a maximum length of the allowed composition is given to ensure that the length of the composition assigned to a trip is not longer than the length of the shortest platform amongst all platforms where the train has a stop. Platform lengths along the network may differ, so different trips may have different maximum composition lengths.

After a trip has been operated the composition is usually assigned to a next *succeeding* trip. Such a combination of two succeeding trips is also called a *connection* (note that this also holds for end stations of a line). It is allowed to first change the composition before it is used on the next trip. However, the transition from one composition to another must follow certain business rules. Depending on the direction in which the trip is operated and the station layout, it is stated on which side of the composition it is allowed to add (couple) extra rolling stock units and on which side it is allowed to decouple units. Such a composition change is called a *transition*.

The possibility of coupling and decoupling units requires that we keep track of the order of the rolling stock units within the composition. Most of the time it is only allowed to (de)couple on one side of the composition. For example, in the Netherlands, if a train continues in the same direction, coupling is usually done at the front of the train. This will speed up the process since the rolling stock unit could, in this situation, already be placed there before the train arrives. Decoupling will most likely take place at the rear of the train if the train continues in the same direction. As a consequence, the train can leave before the decoupled unit is shunted away. Figure 5.1 shows an

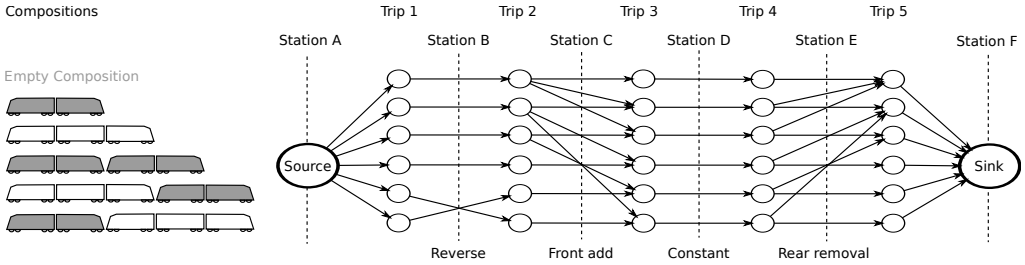


Figure 5.1: An example of a transition network illustrating how different rules affect the possible composition transitions from one trip to the successor trip. On the left the example compositions are represented, and to the right the transition network is depicted. Any path from the source to the sink represents a transition scheme that is feasible. A train travels from one station to the next. The transition rules depend on the station layout and business rules. Examples in the figure are: reversing the composition (it returns in the direction it came from), only coupling units to the front, allowing no change, and only allowing units to be decoupled from the rear.

example of a transition network illustrating how different composition transition rules work. An explanation of the Figure is given in the caption. Keeping track of the order of the units within a composition makes the model more complicated than a simple assignment problem. Furthermore, coupling or decoupling units requires shunting personnel and time, and is therefore penalized in the objective function.

In this paper, we assume that shunting yards associated with certain stations have infinite capacity to accommodate all composition changes and to park decoupled rolling stock units. After decoupling a unit, we require a certain minimum time duration before that unit can be coupled to a new trip in another composition. This restriction reflects the time required to move a decoupled unit to the shunting yard and from the yard back to a platform to be coupled again.

For each trip an estimated passenger demand is given. It is not strictly required that the capacity for passengers of the composition assigned to a trip is equal or larger than the passenger demand for that trip. However, assigning a composition to a trip which cannot accommodate all passengers is penalized.

On the other hand, considering depreciation and energy efficiency, having more capacity than there is demand for a trip is also not preferred. Therefore, the number of kilometers driven by all carriages is penalized as well. A trade-off between the number of seat-shortages and the number of carriage kilometers must be found in the rolling stock circulation.

We must ensure that the next day the rolling stock schedule can be applied as planned. Therefore, the schedule must be such that at the end of the day the rolling stock units are spread over the shunting yards such that at each shunting yard there are as many rolling stock units as are required at the start of the next day (also called *end-of-day balance*). We allow differences in the end-of-day balance, but against a certain penalty, since for each negative unit difference a new deadhead trip should be planned during the night to re-balance the inventories. Deadheading is expensive in practice since it requires additional manpower and causes additional equipment depreciation.

Especially in the case of rescheduling, it is of significant importance that solutions are found very fast. The trains need to keep running, which means that the operator can not wait for one hour to decide how to adapt the resource schedules in order to handle the ongoing disruption.

5.3 Mathematical formulations

In this section we introduce the mathematical formulations of the two solution methods. We present a compact formulation first, termed the Composition Model , which is equivalent to the formulation presented by Nielsen et al. [20] (based on Fioole et al. [14]). A path based (column generation) formulation is presented next, which is an extension of the formulation presented by Haahr et al. [16]. A common notation for both methods is presented before both formulations in Sections 5.3.1 and 5.3.2 respectively.

In Tables 5.1, 5.2 and 5.3, the sets, parameters, and variables are introduced and explained respectively.

Set	Description
$\mathcal{D}$	Set of stations that have associated depot tracks
$\mathcal{C}$	Set of all compositions
$\mathcal{T}$	Set of all trips
$\mathcal{T}_{\leftrightarrow}$	Set of all connections
$\mathbf{s}_r$	The first trip of connection $r \in \mathcal{T}_{\leftrightarrow}$
$\mathbf{t}_r$	The second trip of connection $r \in \mathcal{T}_{\leftrightarrow}$
$\mathbf{d}_r$	The station at which connection $r \in \mathcal{T}_{\leftrightarrow}$ takes place
$\mathcal{U}$	Set of train unit types
$\mathcal{C}_r^2$	Set of all combinations of compositions (c, c') allowed for connection r. This means that composition c is allowed for trip $\mathbf{s}_r$ and composition c' is allowed for composition $\mathbf{t}_r$

Table 5.1: List of sets and elements

Parameter	Description
sod_d^u	Starting station inventory of unit type $u \in \mathcal{U}$ at station $d \in \mathcal{D}$
eod_d^u	Target end-of-day inventory of unit type $u \in \mathcal{U}$ at station $d \in \mathcal{D}$
$compCost_c^t$	Combined costs of trip cancellation, trip seat shortages, and operational costs if composition $c \in \mathcal{C}$ is used on trip $t \in \mathcal{T}$
$transCost_{c,c'}^r$	Coupling costs of changing from composition $c \in \mathcal{C}$ to $c' \in \mathcal{C}$ between trip $\mathbf{s}_r$ and $\mathbf{t}_r$ of connection r
$eodCost$	The penalty for a single end of day balance shortage

Table 5.2: List of parameters and coefficients

Variable	Description
y_c^t	Binary variable deciding whether composition $c \in \mathcal{C}$ is used on trip $t \in \mathcal{T}$
$z_{c,c'}^r$	Binary variable deciding whether composition $c \in \mathcal{C}$ and $c' \in \mathcal{C}$ are used for trip $\mathbf{s}_r$ and $\mathbf{t}_r$ respectively
i_d^u	Integer variable representing the end-of-day balance shortage for unit type $u \in \mathcal{U}$ at station $d \in \mathcal{D}$

Table 5.3: List of Variables

Both formulations are mixed integer linear programs, where the objective function is defined

as:

$$\text{Minimize: } \sum_{t\in\mathcal{T}}\sum_{c\in\mathcal{C}} compCost^t_c \cdot y^t_c \tag{5.1}$$

$$+ \sum_{r\in\mathcal{T}_{\leftrightarrow}}\sum_{c\in\mathcal{C}}\sum_{c'\in\mathcal{C}} transCost^r_{c,c'} \cdot z^r_{c,c'} \tag{5.2}$$

$$+ \sum_{u\in\mathcal{U}}\sum_{d\in\mathcal{D}} eodCost \cdot i^u_d \tag{5.3}$$

The objective function consists of three parts: costs for assigning compositions to trips, costs for assigning transitions between compositions, and costs for having end of day balance shortages. Trip cancellation, seat shortage and operational costs are included in (5.1). The shunting costs are included in (5.2), and the end-of-day shortage costs are accounted for in (5.3).

In both formulations, the relationship between the composition variables and the transition variables is constrained in order to comply with the physical rules and business logic. For a given trip the rules stipulate which compositions are allowed on the following trip:

$$\sum_{c\in\mathcal{C}} y^t_c = 1 \qquad \forall t \in \mathcal{T} \tag{5.4}$$

$$y^{\mathbf{s}_r}_c = \sum_{c'\in\{c_2|(c_1,c_2)\in\mathcal{C}^2_r\wedge c_1=c\}} z^r_{c,c'} \qquad \forall r \in \mathcal{T}_{\leftrightarrow}, c \in \mathcal{C} \tag{5.5}$$

$$y^{\mathbf{t}_r}_{c'} = \sum_{c\in\{c_1|(c_1,c_2)\in\mathcal{C}^2_r\wedge c_2=c'\}} z^r_{c,c'} \qquad \forall r \in \mathcal{T}_{\leftrightarrow}, c' \in \mathcal{C} \tag{5.6}$$

Constraints (5.4) ensure that exactly one composition is assigned to each trip and Constraints (5.5) and (5.6) ensure that a feasible path is found in the transition network, see Figure 5.1. Note, that the empty composition is a valid composition, however, this composition assignment has a high penalty, as it corresponds to canceling a trip. The composition assigned to the incoming trip in a connection ($y^{\mathbf{s}_r}_c$) must be equal to the actual incoming composition in the chosen transition, as modelled by Constraints (5.5). On the other hand, the composition assigned to the outgoing trip in a connection ($y^{\mathbf{t}_r}_{c'}$) must be equal to the actual outgoing composition in the chosen transition, as modelled by Constraints (5.6).

These constraints do not consider the availability of rolling stock units and do not measure the end-of-day shortages i^u_d. In the following two sections we discuss how the two different models take this into account.

5.3.1 The Composition Model

The first option to consider the availability of rolling stock is to keep track of the inventory of rolling stock units at the stations. This option is applied in the formulation of Nielsen et al. [20], which is based on Fioole et al. [14]. In this section this formulation is summarized.

For this formulation additional parameters and variables are necessary to determine at each station the inventory and the number of coupled and decoupled units. The parameters $coup^u_{c,c'}$ and $uncoup^u_{c,c'}$ indicate how many rolling stock units of type $u \in \mathcal{U}$ should be coupled or uncoupled respectively if the composition changes from $c \in \mathcal{C}$ to $c' \in \mathcal{C}$. These values can not be negative. For instance, if 2 units of type u need to be coupled during a composition change from composition c to composition c', then $coup^u_{c,c'} = 2$ and $uncoup^u_{c,c'} = 0$. Furthermore, we assume a certain processing time to shunt a decoupled unit to a shunting yard and to get it back from the shunting yard. Therefore, a parameter τ^-_r is used for each connection r indicating the time it will take before the units that are decoupled during connection $r \in \mathcal{T}_{\leftrightarrow}$ (in between trips $\mathbf{s}_r$ and $\mathbf{t}_r$) are

available again for coupling. Also a parameter τ_r^+ is required for each connection r indicating the time at which connection r takes place. This is the time the units, which need to be coupled to the composition between trips $\mathbf{s}_r$ and $\mathbf{t}_r$, should be available. The station at which connection r takes place is indicated by $\mathbf{d}_r$. Note that this is the station where trip, $\mathbf{s}_r$, ends and where trip, $\mathbf{t}_r$, starts.

For connection $r \in \mathcal{T}_\leftrightarrow$ and rolling stock unit type $u \in \mathcal{U}$, the non-negative integer variables $v_{r,u}^+$ and $v_{r,u}^-$ indicate respectively the number of rolling stock units of type u that are coupled to the composition between trips $\mathbf{s}_r$ and $\mathbf{t}_r$, and the number of rolling stock units of type u that are uncoupled from the composition between trips $\mathbf{s}_r$ and $\mathbf{t}_r$. Furthermore, a variable $inv_{r,u}$, representing the inventory just after connection $r \in \mathcal{T}_\leftrightarrow$ (τ_t^+) of rolling stock units of type $u \in \mathcal{U}$ at station $\mathbf{d}_r$, is required.

Then the following constraints are added to ensure that: *i)* the inventory is non-negative at each station and each time period and *ii)* the end-of-day balance shortage is correctly measured.

$$v_{r,u}^+ = \sum_{(c,c') \in \mathcal{C}_r^2} coup_{c,c'}^u \cdot z_{c,c'}^r \qquad \forall r \in \mathcal{T}_\leftrightarrow, u \in \mathcal{U} \qquad (5.7)$$

$$v_{r,u}^- = \sum_{(c,c') \in \mathcal{C}_r^2} uncoup_{c,c'}^u \cdot z_{c,c'}^r \qquad \forall r \in \mathcal{T}_\leftrightarrow, u \in \mathcal{U} \qquad (5.8)$$

$$inv_{r,u} = sod_{\mathbf{d}_r}^u - \sum_{\substack{r' \in \mathcal{T}_\leftrightarrow: \\ \mathbf{d}_{r'} = \mathbf{d}_r, \\ \tau_{r'}^+ \leq \tau_r^+}} v_{r',u}^+ + \sum_{\substack{r' \in \mathcal{T}_\leftrightarrow: \\ \mathbf{d}_{r'} = \mathbf{d}_r, \\ \tau_{r'}^- \leq \tau_r^+}} v_{r',u}^- \qquad \forall r \in \mathcal{T}_\leftrightarrow, u \in \mathcal{U} \qquad (5.9)$$

$$i_d^u \geq (eod_d^u - sod_d^u) - \sum_{r \in \mathcal{T}_\leftrightarrow: \mathbf{d}_r = d} v_{r',u}^- + \sum_{r \in \mathcal{T}_\leftrightarrow: \mathbf{d}_r = d} v_{r',u}^+ \qquad \forall d \in \mathcal{D}, u \in \mathcal{U} \qquad (5.10)$$

$$i_d^u \in \mathbb{Z}^+ \qquad \forall d \in \mathcal{D}, u \in \mathcal{U} \qquad (5.11)$$

$$v_{r,u}^+, v_{r,u}^-, inv_{r,u} \in \mathbb{R}^+ \qquad \forall r \in \mathcal{T}_\leftrightarrow, u \in \mathcal{U} \qquad (5.12)$$

Constraints (5.9) determine the inventory just after a connection takes place. The inventory is equal to the start inventory at the station, minus all units being coupled to compositions in earlier connections, plus all units which are available again after being decoupled from compositions in earlier connections. Note, that since the variables $inv_{r,u}$ are non-negative it also ensures that the inventory is zero or positive at all times. Constraints (5.10) make sure that the end-of-day shortage is captured. This is equal to the planned difference between the start and end inventory, minus the realized difference between the number of units coupled (departed) and decoupled (arrived) during the day at that station. Constraints (5.11) and (5.12) ensure the non-negativity of the number of (de)couplings, inventories and end-of-day shortages. We note that the integrality of the end-of-day balance variables can be relaxed as these will be naturally integer valued.

5.3.2 The Path Based Model

In Section 5.3.1 a flow based model for the Rolling Stock (Re)Scheduling Problem is presented that identifies the correct number of units flowing from station to station. The Rolling Stock (Re)scheduling Problem can also be formulated using a path based model. In contrast to the flow based model, the path based formulation explicitly considers the route of each individual rolling stock unit and in this way availability of rolling stock is guaranteed. The main advantages of this approach is the ability to model unit-specific constraints more easily, e.g., mileage-limitations before next maintenance, or route-choices for units displaying region-dependent commercials. We refer to [16] for a study using unit specific constraints. In contrast, the main disadvantage is the

added complexity of monitoring the unit routes, in contrast to only considering the flow of unit types in the Composition Model.

The number of possible paths for a rolling stock unit grows exponentially in the number of trips, making the full path model computationally intractable. We therefore propose to solve this method by using column generation, i.e., only a subset of all possible paths is considered. By iteratively solving this reduced model (also known as the master problem) and adding columns with negative reduced cost (found using a so-called subproblem) we are able to get an optimal solution. We refer to Desaulniers et al. [13] for a detailed introduction to column generation and *Branch and Price* frameworks.

Recall from Section 5.1 that this model is an extension of the model presented in Haahr et al. [16]. The model is solved using a *Branch and Price* framework, i.e, a *Branch And Bound* (BAB) approach where columns are added dynamically (as needed to prove optimality) at every node in the BAB search tree. The main difference is that we have to consider the order of the units within a composition. Therefore Constraints (5.4)-(5.6) are used in this formulation as well.

In the remainder of this formulation, $\mathcal{P}_d^u$ is defined as the set of all paths for unit type $u \in \mathcal{U}$ starting in station $d \in \mathcal{D}$. A path describes a chronological list of trips that are performed by a single rolling stock unit, i.e., a unit's schedule for the planning period. The set of all possible paths is thus denoted by $\mathcal{P} := \bigcup_{u \in \mathcal{U}, d \in \mathcal{D}} \mathcal{P}_d^u$. Shunting operations are implicit as a unit has to be uncoupled (and therefore shunted) whenever the unit exits a trip connection, and coupled whenever the unit, going out of the shunting yard, enters a connection.

The (re)scheduling problem simply consists of assigning exactly one path to each vehicle, subject to a number of constraints. Therefore, the master formulation contains additional sets of binary decision parameters and variables. The first set of binary parameters, α_p^t, states for each path $p \in \mathcal{P}$ and trip $t \in \mathcal{T}$ whether path p visits trip t ($\alpha_p^t = 1$) or not ($\alpha_p^t = 0$). The second set of binary parameters, β_p^d, takes the value 1 if path $p \in \mathcal{P}$ terminates at station $d \in \mathcal{D}$, and otherwise equals 0. The third set of integer parameters, μ_c^u, indicates how many units of type $u \in \mathcal{U}$ are assigned to composition $c \in \mathcal{C}$. The binary variables $\lambda_p \in \{0, 1\}$ determine whether path $p \in \mathcal{P}$ is selected in the final solution or not.

We note that the penalties in (5.1) and (5.2) can be included in the subproblem instead of in the master problem, or alternatively partially included in both. Our preliminary results have shown that it is beneficial for the overall column generation convergence to put a part of the coupling costs in the subproblem. Without awareness of the coupling costs in the subproblem it may generate paths that are unnecessarily expensive, and possibly also incompatible in the master problem.

This leads to the following constraints in the master problem:

$$\text{Objective: } (5.1) - (5.3) \tag{5.13}$$

$$\text{Constraints: } (5.4) - (5.6)$$

$$\sum_{d \in D} \sum_{p \in \mathcal{P}_d^u} \alpha_p^t \lambda_p = \sum_{c \in \mathcal{C}} \mu_c^u y_c^t \qquad \forall u \in \mathcal{U}, t \in \mathcal{T} \tag{5.14}$$

$$\sum_{p \in \mathcal{P}_d^u} \lambda_p = sod_d^u \qquad \forall u \in \mathcal{U}, d \in \mathcal{D} \tag{5.15}$$

$$i_d^u + \sum_{p \in \mathcal{P}_d^u} \beta_p^d \lambda_p \geq eod_d^u \qquad \forall u \in \mathcal{U}, d \in \mathcal{D} \tag{5.16}$$

$$\lambda_p \in \{0, 1\}, \quad y_c^t \in \{0, 1\}, \quad i_d^u \in \mathbb{Z}_0^+ \tag{5.17}$$

Without equations (5.4)-(5.6) the master problem solely consists of finding a set of rolling stock paths. Together with Constraints (5.14) these constraints ensure that only feasible compositions and composition transitions are made.

In comparison with Nielsen et al. [20] the path-formulation does not consider inventories at the stations, but it models individual paths for rolling stock units, thereby implicitly handling the inventories. The composition and path variables are linked by Constraints (5.14) to ensure that the correct number of rolling stock units (and types) are assigned to each trip composition. Constraints (5.15) ensure that exactly one path is assigned to every rolling stock unit. Note that it is necessary for all units to have a path in order to account for the end-of-day balance. A unit's path contains no trips if the unit stays at a depot for the entire period. The end-of-day balance is enforced in Constraints (5.16). A slack (which is penalized in the objective) is inflicted if insufficient units terminate at respective stations. Finally, the domains of the variables are shown in (5.17). We note that the integrality of the end-of-day balance variables can be relaxed as these will be naturally integer valued.

The master problem ((5.1)-(5.6) and (5.14)-(5.17)) is solved iteratively while adding new columns (path variables) that have negative reduced cost. A subproblem is solved, using the duals of (5.14)-(5.16), to find such columns or to prove that they do not exist. This problem can be solved as a shortest path problem (or with resource constraints when unit-specific constraints are enforced). When no columns exist with negative reduced cost we have solved the LP relaxation to optimality.

A *Branch and Price* framework is required to solve the problem, as the LP relaxations of the master problem are not necessarily integral. The framework setup and branching rules described by Haahr et al. [16] are adopted, where branches can be made that stipulate that the number of units on one trip is integral, and that the number of units originating or ending in a station is integral. In addition to these branching rules, we also introduce a new type of branching on the composition variables. For any given trip, an optimal LP solution is not required to assign a value of 1 to one of the composition variables, but can assign a fractional value to some of them as long as the sum is 1. In such cases the branching rule partitions the trip's composition variables into two groups, such that the sum of the fractional values is non-zero in both groups. Note that this is stronger than finding one variable to branch on. Given a trip t with multiple non-zero composition assignments in an LP relaxed solution, we obtain two branches:

$$\sum_{c \in C_1} y_c^t \geq 1$$

and

$$\sum_{c \in C_2} y_c^t \geq 1$$

where C_1 and C_2 define the described partitions. In each branch the sum of one group should be greater than or equal to one. Consequently the sum of the other group is then zero. The same reasoning holds in the other branch.

Subproblem Graph Example

An example of the subproblem graph is shown in Figure 5.2. Note that the underlying graph is acyclic and the weights can be negative. We refer to Haahr et al. [16] for an in-depth description. Note, that the structure of the subproblem is the same as in Haahr et al. [16] even though we take compositions into account in the overall method. The example contains three stations whose events (departures and arrivals) are shown as vertically aligned vertices. Stations with associated depot tracks have a paired (gray color) node, representing the depot (i.e. shunting yard). The graph contains one source (S) that has one out-going arc to the first events of all stations, and one target node (T) that has one in-going arc for the last events on different stations. Intermediate arcs represent train movements (moving units between the depot tracks and a platform) and trips travelling from one station to another. Mileage costs are set on trip arcs and coupling costs are set on shunting arcs. Dual costs from the linking Constraints (5.14) are assigned to the trip arcs.

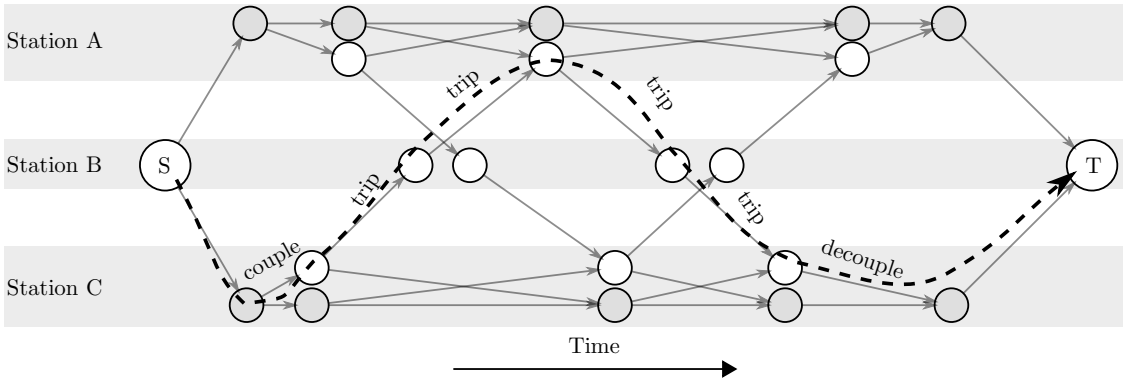


Figure 5.2: A simplified example of the acyclic time-space network showing some of the possible coupling arcs. Three stations (A,B and C) are illustrated where only two of them (A and C) have associated depot tracks. An example of a feasible unit schedule has been highlighted using a dashed path.

Inventory duals from Constraints (5.15) are assigned to the arcs extending from the source. Finally, duals from Constraints (5.16), on the end-of-day balances, are assigned to the target node arcs. We note, that the subproblem can without loss of generality be changed slightly in implementation as some nodes and edges can be altered or removed.

Alternative Formulations

An equivalent mixed integer program can be obtained by replacing the composition transition rules (Constraints (5.5)-(5.6)) with constraints based on the composition variables. The resulting formulation has fewer constraints, which may provide a significant speedup when solving the LP relaxations. A column generation framework relies on solving the LP relaxation many times. For any pair of successive trips say t_1 and t_2 (i.e. a connection) we would require:

$$y_c^{t_1} \leq \sum_{c' \in C_c^3} y_{c'}^{t_2} \quad \forall c \in \mathcal{C}$$

Where C_c^3 is the set of allowed compositions in t_2 following the composition c on trip t_1. These constraints limit the origin composition c, for example $y_c^{t_1}$ cannot be set to 1 if the following trip has an incompatible composition. Likewise, a similar set of constraints can be defined that limit the target composition. Preliminary results show that this alternative formulation weakens the LP relaxation too drastically; the relaxed solutions are more fractional. The benefit of faster LP solution times does not outweigh the weaker relaxation in general.

Another equivalent formulation can be obtained by modeling the composition transition rules with conflicting path constraints. Given a full enumeration of all possible paths in the formulation presented in Haahr et al. [16] (which routes all rolling stock units without any restrictions on train compositions), the composition rules can be enforced by including constraints that prohibit the selection of pairs of conflicting routes:

$$\lambda_{p_1} + \lambda_{p_2} \leq 1 \quad \forall (p_1, p_2) \in \Pi$$

where Π now denotes the set of all conflicting paths. Several important composition rules can be modeled using such pairwise conflict pairs, e.g., only allowing a unit to couple or uncouple on one end of the existing train composition, or disallowing incompatible unit types to be combined on train services.

For example, consider a sequence of trips { a, b, c, d }. Say path p_1 services all trips, path p_2 only services trips {b, c}, and path p_3 only services trips {b, c, d}. Assume that the station between

a and b only allows units to couple and uncouple at the front of the train, and that the station between c and d only allow units to couple or uncouple at the back of the train. As a consequence, paths p_1 and p_2 cannot coexist, as the latter path cannot be coupled to the front of the train and uncoupled from the back. Thus $(p_1, p_2) \in \Pi$. On the other hand, p_1 and p_2 are both compatible with p_3.

The obvious disadvantage of the formulation is the potentially large number of resulting conflict constraints. A delayed cut callback routine, that generates the conflicting constraint as they become violated, can be adopted in a Branch-and-Cut method, and possible remedy this drawback as only a few paths are expected to enter the Simplex basis. However, the non-static row-dimension of the formulation complicates an exact delayed column generation as there is no obvious effective solution approach for the resulting subproblem, where new variables with negative reduced-costs must be found or proven to not exist. Without formally knowing all the possible conflict constraints in advance, it is non-trivial to formulate a subproblem that can account for all necessary dual contributions.

If a full enumeration of all paths is intractable, then a heuristic framework can be adopted that pre-generates an initial set of paths. When all paths are known in advance, the set of pairwise conflicting paths can be deduced and inserted into the formulation. A MIP solver can then be used to solve the resulting formulation. A disadvantage of this approach is the generation of the initial pool of paths. A set of initial paths must be generated, which must contain good and non-conflicting paths. Some of the paths must be able to contain overlapping trips without being in conflict. One approach can include all paths generated while solving the LP-relaxation of the root node in the formulation presented by Haahr et al. [16]. The resulting paths are not proven to be compatible, but this approach requires no special phase for selecting initial paths.

5.3.3 Delayed Transition Constraint Generation

Preliminary results show that the LP relaxation of the formulation requires a significant amount of pivot operations to solve. This is highly undesirable since we are continuously solving the formulation after adding columns in every iteration of the column generation process. We discovered that solving the linear relaxation of the master problem with the Dual Simplex Method was significantly faster than using the Primal Simplex Method. However, in column generation we unfortunately rely on the Primal Simplex Method as subsequent LP resolves can be hot-started using the old optimal solution after adding new columns.

These observations did however lead us to experiment with the formulation in order to improve convergence or runtime speed of the linear relaxation. Constraints (5.5) and (5.6) not only represent a significant amount of constraints in the presented formulation, but also demand a rather restricted set of columns in order to remain feasible during the convergence process. As a result, unnecessary columns are generated making the approach slow. When converging against the optimal linear relaxation, these constraints are unnecessarily restrictive as the intermediate set of states (fractional LP solutions) is not important. By removing these constraints from consideration initially, the advantages are three-fold: we achieve a smaller linear program to resolve, we may gain a faster convergence, and finally we get better duals due to the omission of equality constraints. However, this extension comes at the cost of more added complexity. In the method we therefore propose a variant that dynamically adds Constraints (5.5) and (5.6) as they become violated once the linear program has converged. This step merely consists of looping through the allowed composition transitions and checking whether the current solution violates any constraint.

For the sake of simplicity we only add the dynamic constraints once no negative reduced cost columns can be found. Thus, the proposed variant searches for new columns first, and for violated constraints second. Note, when adding new constraints, the framework searches for negative reduced cost columns again since a new constraint changes the solution. The LP model has

been solved to optimality once no columns can be generated and none of the constraints are violated.

5.4 Computational Experiments

In this section we present and discuss our computational experiments. We test both models on a scheduling case from Netherlands Railways (NS) and on a scheduling case from the Copenhagen Suburban Railway Operator (DSB S-tog). Additionally, we test the models on different rescheduling cases from both NS and DSB S-tog. In this way we thus make a comparison between the (re)scheduling approaches.

We start with describing the rolling stock scheduling instances of both networks and introduce the parameter settings we have adopted. These parameter settings are chosen after several runs to benchmark their influence. Thereafter, a comparison of the computational results of the scheduling instances is made. Then, the rescheduling instances are described in detail for both the NS and DSB S-tog instances. Finally, we conclude the section with an overview of the different rescheduling results.

5.4.1 Rolling Stock Scheduling

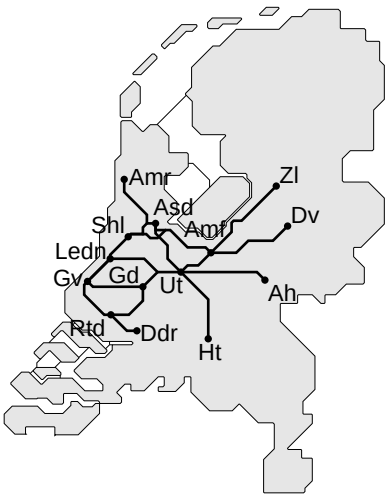
Netherlands Railways

A single instance on the Dutch railway network that spans the major part of the intercity network of NS is selected as computational case. Figure 5.3 shows the trajectory of the network. The majority of the lines are in the western part of the Netherlands which is the busiest part of the Dutch railway network. In total we consider the timetable of 16 distinct lines of a certain weekday, namely the Monday. There are different timetables required on the Friday and in the weekend. However, the differences between those timetables are small, while the differences in passenger demand could be large. Therefore, we only consider the timetable of the Monday as scheduling instance. Note that all trips of the 16 lines during the day are taken into account. In general the lines are operated half-hourly, but some are operated on an hourly basis. This leads to a total of 2324 trips between 14 different major stations. There are many trips between stations where no shunting is allowed. As a consequence, the compositions of those trips are the same as the ones appointed to their predecessor trips. We use a preprocess step to merge those trips. After preprocessing the total number of non-reducible trips is 727, i.e., in fact only 727 trip compositions need to be decided as the rest will be fixed due to the composition transition rules.

We have two different rolling stock types available. A rolling stock unit of type a consists of 4 carriages and has a passenger capacity of 405 seats, and a rolling stock unit of type b consists of 6 carriages and has a passenger capacity of 597 seats. The maximum composition length for all trips is 14 carriages in the considered network. This leads in total to 11 different compositions that can be appointed to a trip: a, aa, aaa, b, bb, ab, ba, aab, aba, baa, and the empty composition meaning that a trip is cancelled.

In the scheduling phase of NS the goal is to determine the start and end inventory at every station, and to appoint compositions to trips to fulfill the passenger demand. This is solved in two steps. First a suitable initial fleet distribution is found. Afterwards the found initial inventory setting is used to create the rolling stock circulation.

In the first step, our objective is to create a rolling stock circulation that covers all demand while using the least amount of carriages. To this end, the model determines the number of rolling stock units required at the start of the day at each station to create such a circulation. We set a small penalty on the number of carriage kilometers. No penalty at all is set on the number of shunting



Line	Stations	Frequency
500	Gv Gd Ut Amf Zl	hourly
700	Shl Amf Zl	hourly
800	Amr Zd Asd Ut Ht	half hourly
1500	Asd Amf Dv	half hourly
1600	Shl Amf Dv	hourly
1700	Gv Gd Ut Amf Dv	hourly
1900	Gv Rtd Ddr	half hourly
2000	Gv Gd Ut Ah	half hourly
2100	Asd Shl Ledn Gv Rtd Ddr	half hourly
2600	Asd Shl Ledn Gv	half hourly
2800	Rtd Gd Ut Amf	half hourly
3000	Amr Asd Ut Ah	half hourly
3500	Shl Ut Ht	half hourly
8800	Ledn Ut	half hourly
20500	Rtd Gd Ut	hourly
21700	Rtd Gd Ut	hourly

Figure 5.3: The NS network considered in the test instances.

operations applied during the day, and no penalty is given for deviations between the start and end inventory.

Our first step provides us with a start circulation that contains in total 113 units of type a and 58 units of type b. This start circulation is able to cover all passenger demand.

In the second step, we use the start inventory created in the first step as fixed input. It is not allowed to deviate from this start inventory and the objective is then to find a rolling stock circulation that minimizes the costs using the coefficients as shown in Table 5.4. These penalty values are commonly used in literature, but we check the influence of changing the Mileage and Seat Shortage penalty ratio on the results in Section 5.4.1.

Here, *Cancel* means the penalty for cancelling a trip, *Mileage* denotes the penalty for driving a single carriage over one kilometer, *Seat Shortage* defines the penalty for having a seat shortage per passenger shortage per kilometer, *Shunting* refers to the penalty for doing a shunting operation, and, finally, *End diff* denotes the penalty for a negative difference between the preferred end of day inventory and the actual end inventory at a station. Results are shown and discussed in Section 5.4.1.

Coefficient	NS
Cancel	1 000 000
Mileage	1
Seat Shortage	0.5
Shunting	1 000
End diff	10 000

Table 5.4: Penalty values used for the NS scheduling case

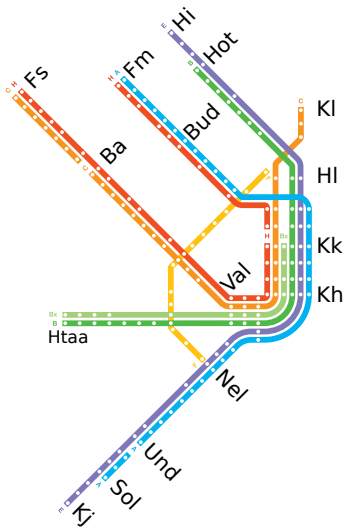
DSB S-tog

The suburban train service network provided by DSB S-tog in Copenhagen is operated using a weekly periodic timetable and an hourly periodic timetable on a daily basis. Weekdays and

weekends are operated using slightly different timetables, and an additional set of night trains are operated on Friday and Saturday nights. This leaves a total of four different timetable schedules: Monday (to Thursday), Friday, Saturday and Sunday. Table 5.5 summarizes the instance specifics. Note that the A, B, C, and F lines are considered in all cases, while the Monday and Friday instances contain more lines than just these 4.

Name	Stops	Trips	Trips*	Weekday	Lines
DSBfri	28 719	4 558	886	Friday	A,B,Bx,C,E,F&H
DSBsat	20 474	1 916	590	Saturday	A,B,C&F
DSBsun	19 919	1 871	574	Sunday	A,B,C&F
DSBmon	28 017	4 468	868	Monday	A,B,Bx,C,E,F&H

Table 5.5: Four timetables operated by DSB S-tog. The columns respectively show the instance names, total number of stops, total number of trips, total number of non-reducible trips (Trips*), weekday, and finally the lines that are running.



Line	Stations	Frequency	
A	Kj Sol Und Kh Kk Bud Fm	10&20	20
B	Htaa Val Kh Kk Hl Hot	10&20	20
C	Fs Ba Val Kh Kk Hl Kl	10&20	20
E	Kj Sol Und Kh Kk Hl Hot Hi	10&20	20
F	(Kl) Hl Nel	10&20	20
H	Fs Ba Val Kh Kk Bud Fm	20	20

Figure 5.4: The DSB S-tog network considered in the test instances. Only key stations are listed in the table. The table columns show line name, station name and the operated frequencies in minutes for Monday-Friday and Saturday-Sunday respectively.

Two unit types are used to perform all train services. The smaller unit type e is 42 meters long and contains 150 passenger seats, the larger unit type f is 83 meters and holds 336 passenger seats. Platform lengths vary from station to station in Denmark as well, and the current lines partition the composition lengths into two sets. The first set allows only small compositions e, ee, and f. The second set also allows larger compositions eee, fe, ef, and ff.

We use distinct parameter settings for NS and S-tog since the characteristics of the network and train unit fleets are different. The parameter settings for the S-tog network can be found in Table 5.6. As can be seen, the only difference between the two settings is the mileage penalty. This is, among other factors, because the rolling stock units of DSB S-tog are measured by length in meters in contrast to the number of carriages. The length in meters is substantially larger than the number of carriages.

DSB S-tog provided us with a start inventory. Thus, for S-tog instances it is not necessary to perform the first step of scheduling, that was described for NS instances. The start inventory contains 31 units of type e and 103 units of type f.

Coefficient	DSB S-tog
Cancel	1 000 000
Mileage	0.1
Seat Shortage	0.5
Shunting	1 000
Start end diff	10 000

Table 5.6: Penalty values used for the DSB S-tog scheduling case

Computational results

In the scheduling phase we determine the start inventory at every station, the end inventory at every station, and we appoint compositions to trips to transport passengers. For the NS case, the first phase as explained in Section 5.4.1, to create a start inventory, is solved within seconds and the start inventory is fixed from here on. Note that the end inventory is not fixed, but we penalize for deviations with respect to the start inventory, assuming that the next day the same schedule needs to be operated.

In this section we benchmark the two proposed methods for solving the rolling stock scheduling problem on both NS and DSB S-tog instances. We compare both NS and DSB S-tog instances on the second phase of scheduling. The mathematical formulations are equivalent and we have verified that the optimal objective costs are identical for the two methods. The objective function is a mix of several penalties where the exact balance between them is non-trivial to set. Due to this, there is little justification for solving to true optimality. In practice, it might be enough to accept solutions that are within 1% of optimality, but to be able to compare the approaches accurately we solve to optimality.

In order to justify the parameter values as presented in Tables 5.4 and 5.5, we investigated the trade-off between the seat shortage and mileage costs. The results are shown in Figure 5.5. The results show a steady increase in mileage costs as the seat cover moves towards 100%. However, when we get close to 99% we observe diminishing returns. A cover of 100% is preferred by rolling stock planners, however not at all cost. In practice a seat cover close to 100% with a rolling stock circulation that is not overly expensive is chosen.

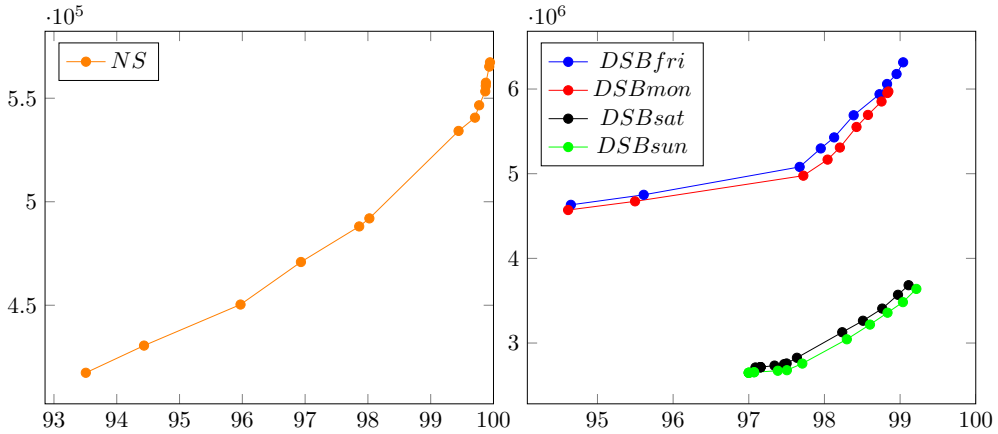


Figure 5.5: Trade-off between mileage and seat/demand cover for the NS (left) and DSB S-tog (right) instance. The vertical and horizontal axis respectively show the mileage usage (in km) and seat cover percentage.

The settings of the parameters do not only influence the costs of the rolling stock circulation,

but also the computation time to find the circulation. Information about how the computation times are affected by the different penalties for seat shortages are provided in Table 5.7 for all scheduling instances. Here A denotes the Path Based Model without row generation, B the Path Based Model with row generation, and N the Composition Model. As can be seen, B is able to solve the scheduling instances on average faster than A. This will be discussed in more detail in Section 5.4.3. Furthermore, the Composition Model is able to solve the instances even faster, but all solution times are acceptable for the planning phase. We assume that a solution method resulting in optimal schedules within 15 minutes can be used in practice.

	NS			DSBmon			DSBfri			DSBsat			DSBsun		
Costs	A	B	N	A	B	N	A	B	N	A	B	N	A	B	N
0.1	444	84	27	398	30	14	314	63	14	14	3	2	21	3	3
0.2	545	65	23	575	44	13	423	59	13	21	6	3	29	4	2
0.3	742	274	16	652	345	21	556	48	21	31	6	3	40	4	3
0.4	712	493	17	579	96	14	479	38	12	57	5	4	41	4	4
0.5	709	465	19	653	119	19	664	37	12	58	10	7	35	4	2
0.6	946	781	16	760	47	11	709	40	14	36	8	5	31	4	3

Table 5.7: Detailed runtimes using the solution methods on the non-disrupted test instances. The columns respectively show the computation times to solve the instances with A, B, and N method.

Recall that the Composition Model and the Path Based Model give the same results in terms of objective values. We have selected an appropriate balance in the following benchmarks by choosing the settings as shown in Table 5.4 and Table 5.6. These settings lead to the results as shown in Table 5.8. We have left out the results of the A method, because they are all dominated by the B method. Note that in the DSB S-tog cases it is not possible to reach a seat coverage equal to 100%. The seat coverage for all instances is above 98%, so almost all passengers fit in the appointed compositions with a seat.

Instance	Obj	MC	SSC	SC	δ	Cover	B (s)	Columns	N (s)
NS	639 065	553 310	15 755	70 000	0	99.9%	465	14 340	19
DSBmon	719 184	555 970	132 214	31 000	0	98.5%	119	5 937	19
DSBfri	727 159	583 505	119 654	24 000	0	98.6%	37	3 627	12
DSBsat	418 148	313 469	87 679	17 000	0	98.3%	10	1 052	7
DSBsun	413 062	297 574	93 489	22 000	0	98.1%	4	1 266	2

Table 5.8: Scheduling Results. The columns respectively show the solved instance, the objective value, the mileage kilometer costs, the overall seat-shortage kilometer costs, the costs for performing shunting operations, the costs for negative differences between start and end inventories, the total seat cover percentage, computation times and columns generated using the B method, and finally the computation times for the N method.

5.4.2 Rolling Stock Rescheduling

In this section we discuss the computational experiments for the rescheduling instances. Note that for the rescheduling instances we assume the demand to be the same as in the original situation. See, for instance, Kroon et al. [18] for a paper that takes dynamic passenger flows into account by means of a simulation step in the Rolling Stock Rescheduling Problem. This aspect is outside the scope of the current paper. In this section, we first give a brief overview of the different rescheduling cases for Netherlands Railways (Section 5.4.2) and DSB S-tog (Section 5.4.2). Thereafter, the computational results are discussed (Section 5.4.2).

Netherlands Railways

In the following we compare both models on different disruption scenarios from NS. Consequently, we have chosen 12 different scenarios, of which an overview is given in Table 5.9. The disruptions take place on the main parts of the network, causing one track to be blocked for at least three hours. This track is blocked in both directions, so there is no railway traffic possible there. The parameter settings used for the NS rescheduling instances are given in Table 5.10. The

Case	Disruped area	Time interval		
1, 2, 3	Gd - Ut	07:00-10:00	16:00-19:00	11:00-15:00
4, 5	Rtd - Gv	16:00-19:00	11:00-15:00	
6	Ledn - Ut	16:00-19:00		
7, 8	Amf - Ut	16:00-19:00	11:00-15:00	
9, 10	Gv - Ledn	16:00-19:00	11:00-15:00	
11, 12	Asd - Ut	16:00-19:00	11:00-15:00	

Table 5.9: Different disruption scenarios NS. The columns show the case numbers, the location of the disruption, and the affected time-slots.

largest penalty is still on cancelling a trip, this is the most important criterion. The penalties for carriage kilometers and seat-shortage kilometers are the same as in the scheduling case. Three new penalties are used in the rescheduling cases: *Shunting Unplanned*, *Shunting Cancelled*, and *End Diff*. Shunting Unplanned denotes the cost for doing a shunting operation that was not originally scheduled. Every new (un)coupling action requires shunting crew members to perform them. As a consequence, the shunting crew needs to be rescheduled. So, we want to minimize the number of additional shunting movements. On the other hand, Shunting Cancelled stands for the penalty for cancelling a scheduled shunting movement. Shunting crew members are instructed to do planned shunting movements during the day, cancelling such a shunting movement means that the tasks for those crew members have to be cancelled. This has to be communicated to the involved crew members. Therefore, we want to minimize the number of cancelled shunting operations as well. However, cancelling shunting movements is preferred over adding additional shunting movements, because arranging new shunting crew members for a task is usually more difficult than cancelling a shunting crew task. Finally, End Diff denotes the penalty for having a negative deviation from the scheduled end of day balance. We leave this coefficient identical to the one used in the planned instance for the start-end-of-day deviation.

Coefficient	NS	DSB S-tog
Cancel	1 000 000	1 000 000
Mileage	1	0.1
Seat Shortage	0.5	0.5
Shunting Unplanned	1 000	1 000
Shunting Cancelled	100	100
End Diff	10 000	10 000

Table 5.10: Penalty values used for the NS and S-tog rescheduling cases.

DSB S-tog

For the DSB S-tog instances, there are also 12 different disruption scenarios, see Table 5.11. These disruptions occur either on the Monday or Friday instances, leading to 24 instances in total.

Case	Disruped Area	Time interval		
1, 2, 3	Val-Fs	15:00-16:00	15:00-19:00	11:00-12:00
4, 5, 6	Val-Fs	11:00-15:00	09:00-13:00	09:00-10:00
7, 8, 9	Kh-Kk	15:00-16:00	15:00-19:00	11:00-12:00
10, 11, 12	Kh-Kk	11:00-15:00	09:00-13:00	09:00-10:00

Table 5.11: Different disruption scenarios DSB S-tog. The columns denote the different case numbers, the location of the disruption, and the affected time-slots.

The same objective coefficients are used as in the NS rescheduling instances. The only difference is the mileage penalty, which is 0.1 instead of 1.0, see Section 5.4.1.

Computational Results

All three approaches give the same results after rescheduling the disrupted instances. Figure 5.6 gives an overview of the solution costs for the rescheduling results. Here, the objective value, the mileage costs, and the seat-shortage costs are displayed per rescheduled instance. The other costs are not mentioned, because they are very small in comparison. The objective value is largest in the cases where trips are cancelled. In the other cases, the objective value is close to the mileage costs. So, the mileage costs are much larger than the seat-shortage costs in all cases. This can also be noted by Figure 5.7, where the total seat coverage percentage per rescheduling instance is shown. The seat coverage is always between 97% and 99.9%, which means that only few passengers do not get a seat in the train.

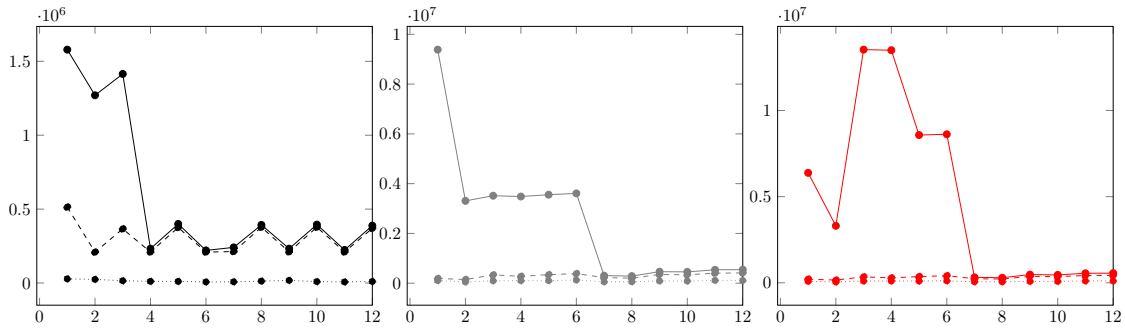


Figure 5.6: Results for the NS (left) and DSB S-tog (Monday middle, Friday right) rescheduling instances. A straight line denotes the objective value, a dashed line the mileage costs, and a dotted line the seat shortage costs. The horizontal axis denotes the case numbers and the vertical axis the values.

Next to finding good results, the computation time is of importance as well. In a disrupted situation, little time is available to reschedule the rolling stock. Therefore, we will compare our models based on their computation times. In Table 5.12 an overview of all computation times required to solve the different instances with the different models is shown. First of all, the solution method B is substantially faster than the solution method A. However, both models are considerably slower than the Composition Model. On the other hand, as explained before, the Path Based Model has other advantages over the Composition Model.

We note that the computation times are influenced by the start time of the disruption - a late disruption involves fewer trips than an earlier disruption, thus a late disruption is easier to solve. In conclusion, both models B and N are applicable for usage in practice. They are able to reschedule the rolling stock fast enough such that the rescheduled circulation can be used by practitioners in

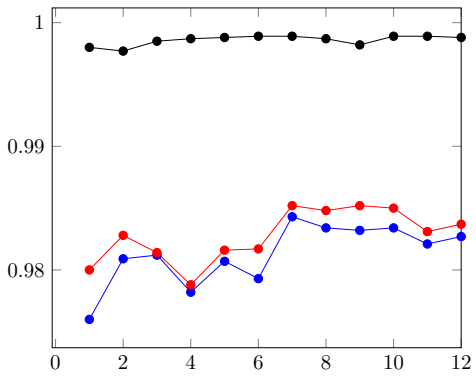


Figure 5.7: The seat-coverage for the rescheduling cases. The black line is for the Dutch instances, the blue line for the Friday DSB S-Tog instances, and the red line for the Monday DSB S-Tog instances. The horizontal axis denotes the case numbers and the vertical axis the seat cover percentage, where 1 stands for 100%.

real-time. On the other hand too high computation times are observed in some cases for the A approach.

	NS			DSBmon			DSBfri		
Case	A	B	N	A	B	N	A	B	N
1	722	269	5	38	22	4	18	7	3
2	20	6	2	4	2	2	5	2	2
3	138	23	5	72	15	5	113	17	5
4	11	3	1	63	23	5	51	7	3
5	133	34	3	176	31	4	151	22	6
6	14	2	1	286	207	8	303	40	7
7	13	13	2	8	2	1	13	4	2
8	159	39	3	6	2	2	9	2	2
9	10	2	1	62	6	3	96	10	4
10	157	26	3	106	18	4	103	8	3
11	15	2	1	222	17	7	284	15	4
12	163	35	3	156	9	3	235	15	6
Average	130	38	3	100	29	4	115	12	4

Table 5.12: Computation time (in seconds) for solving the disruption instances.

5.4.3 Delayed Transition Constraint Generation

Finally, in this section we present details of the average computation time and average number of columns and rows of the A and B methods in Table 5.13. The results show that the dynamic row generation B method is on average around 5 times faster than the normal column generation method A.

Interestingly, the table shows that, on average, the number of generated columns decreases in B for the DSB cases, but increases slightly for the NS cases. However, we observe a consistent decrease in the number of rows.

	A			*B*			*N*
Case	Time	Columns	Rows	Time	Columns	Rows	Time
NS	190	6 178	7 214	51	6 434	3 573	2
DSBmon	143	7 321	6 523	31	4 701	5 677	4
DSBfri	161	7 040	6 891	16	4 373	4 704	4

Table 5.13: The columns show computational time (in seconds), number of columns and rows generated for solving planning and disruption instances. Each row shows the average over 1 planning and 12 disruption cases.

Although B often results in fewer generated columns and rows, there are cases (considering the non-aggregated results) where this is not the case. There exist cases where runtime is improved even if more columns and rows are generated in total - we believe this is due to the better convergence property of B, see Section 5.3.2.

5.5 Conclusion

In this paper a comparison is made between two rolling stock (re)scheduling models by testing them on data sets from two countries (the Netherlands and Denmark). The results show that the considered approaches are not limited or tailored to specific networks. Furthermore, it is the first time that the Composition Model is tested on instances of the DSB S-tog network in Denmark and the Path Based Model on instances from Netherlands Railways.

A sensitivity analysis of the two models is carried out to determine reasonable values for the mileage and seat-shortage penalties for the train services. The results demonstrate that a higher seat-cover requires significantly more kilometers to be carried out by the carriages. For the DSB S-tog case it was, with the different penalties we have tested, not possible to provide a seat for all passengers. This is due to the fact that the passenger demand is sometimes larger than the capacity of the largest composition we can appoint.

In the current tests the column generation approach of the Path Based Model has longer computation times than the Composition Model. However, we believe that the column generation approach can be quite interesting, especially when we want to include unit specific constraints, e.g. due to maintenance.

Next to the scheduling instances, we have tested both approaches on rescheduling instances in the Netherlands and Denmark as well. Disruption instances are smaller than the planning counterparts since all trip decisions up to the start of the disruption are fixed. So in some sense they are easier. However, since we also want to minimize the deviations from the original plan, they are also harder. The experiments show that both models are able to reschedule the rolling circulation fast enough to be used in real-time in both countries.

In future research we want to consider unit specific constraints for the rolling stock, in both a planning and a disruption context. Examples of interesting additional constraints are maintenance appointments and minor rolling stock defects. In the former case units have to reach a maintenance facility in time for an appointment, and in the latter case units with a small defect cannot run on their own but only when coupled to another unit.

Bibliography

[1] R. K. Ahuja, J. Liu, J. B. Orlin, D. Sharma, and L. A. Shughart. Solving real-life locomotive-scheduling problems. *Transportation Science*, 39(4):503–517, 2005.

[2] R.K. Ahuja, T. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall Englewood Cliffs, 1993.

[3] A. Alfieri, R. Groot, L. Kroon, and A. Schrijver. Efficient circulation of railway rolling stock. *Transportation Science*, 40(3):378–391, 2006.

[4] R. Borndörfer, M. Reuther, T. Schlechte, and S. Weider. A hypergraph model for railway vehicle rotation planning. *Proceedings in ATMOS*, pages 146–155, 2011.

[5] B. Bouzaiene-Ayari, C. Cheng, S. Das, R. Fiorillo, and W. B. Powell. From single commodity to multiattribute models for locomotive optimization: A comparison of optimal integer programming and approximate dynamic programming. *To appear in: Transportation Science*, 2014.

[6] P. Brucker, J. Hurink, and T. Rolfes. Routing of railway carriages. *Journal of Global Optimization*, 27(2-3):313–332, 2003.

[7] V. Cacchiani, A. Caprara, and P. Toth. Solving a real-world train-unit assignment problem. *Mathematical Programming*, 124(1-2):207–231, 2010.

[8] V. Cacchiani, D. Huisman, M. Kidd, L.G. Kroon, P. Toth, L. Veelenturf, and J.C. Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, 63:15–37, 2014.

[9] L. Cadarso and Á. Marín. Robust rolling stock in rapid transit networks. *Computers & Operations Research*, 38(8):1131 – 1142, 2011.

[10] L. Cadarso and Á. Marín. Improving robustness of rolling stock circulations in rapid transit networks. *Computers & Operations Research*, 51:146 – 159, 2014.

[11] J.-F. Cordeau, G. Desaulniers, N. Lingaya, F. Soumis, and J. Desrosiers. Simultaneous locomotive and car assignment at VIA Rail Canada. *Transportation Research Part B*, 35(8):767–787, 2002.

[12] J.-F. Cordeau, F. Soumis, and J. Desrosiers. Simultaneous assignment of locomotives and cars to passenger trains. *Operations Research*, 49(4):531–548, 2001.

[13] G. Desaulniers, J. Desrosiers, and M. M Solomon. *Column generation*, volume 5. Springer, 2005.

[14] P.J. Fioole, L.G. Kroon, G. Maróti, and A. Schrijver. A rolling stock circulation model for combining and splitting of passenger trains. *European Journal of Operational Research*, 174(2):1281–1297, 2006.

[15] A. Fügenschuh, H. Homfeld, A. Huck, A. Martin, and Z. Yuan. Scheduling locomotives and car transfers in freight transport. *Transportation Science*, 42(4):478–491, 2008.

[16] J.T. Haahr, R.M. Lusby, J. Larsen, and D. Pisinger. A branch-and-price framework for railway rolling stock rescheduling during disruptions. *DTU Management engineering*, 27p, 2014.

[17] J. Jespersen-Groth, D. Potthoff, J. Clausen, D. Huisman, L. Kroon, G. Maróti, and M. N. Nielsen. *Disruption management in passenger railway transportation.* Springer, 2009.

[18] L. Kroon, G. Maroti, and L. Nielsen. Rescheduling of railway rolling stock with dynamic passenger flows. *Transportation Science*, 2014.

[19] N. Lingaya, J.-F. Cordeau, G. Desaulniers, J. Desrosiers, and F. Soumis. Operational car assignment at {VIA} rail canada. *Transportation Research Part B: Methodological*, 36(9):755 – 778, 2002.

[20] L.K. Nielsen, L.G. Kroon, and G. Maróti. A rolling horizon approach for disruption management of railway rolling stock. *European Journal of Operational Research*, 220(2):496–509, 2012.

[21] M. Peeters and L. Kroon. Circulation of railway rolling stock: a branch-and-price approach. *Computers & Operations Research*, 35(2):538 – 556, 2008.

[22] F. Piu and M. G. Speranza. The locomotive assignment problem: a survey on optimization models. *International Transactions in Operational Research*, 21(3):327–352, 2014.

[23] S. Rouillon, G. Desaulniers, and F. Soumis. An extended branch-and-bound method for locomotive assignment. *Transportation Research Part B: Methodological*, 40(5):404 – 423, 2006.

[24] B. Vaidyanathan, R. K. Ahuja, and J. B. Orlin. The locomotive routing problem. *Transportation Science*, 42(4):492–507, 2008.

[25] J.C. Wagenaar, L.G. Kroon, and M. Schmidt. Maintenance in railway rolling stock rescheduling for passenger railways. *ERIM report series research in management Erasmus Research Institute of Management*, ERS-2015-002-LIS, 2015.

Chapter 6

A Comparison of Optimization Methods for Solving the Depot Matching and Parking Problem

Jørgen T. Haahr* Richard M. Lusby* Joris C. Wagenaar†

*Department of Management Engineering, Technical University of Denmark,
Produktionstorvet, Building 426, DK-2800 Kgs. Lyngby, Denmark
jhaa@dtu.dk, rml@dtu.dk

†Rotterdam School of Management, Erasmus University,
PO Box 1738, 3000 DR Rotterdam, The Netherlands
jwagenaar@rsm.nl

[1] **Abstract** We consider the Train Unit Shunting Problem, an important planning problem for passenger railway operators. This problem entails assigning physical train units to scheduled train services in such a way that the resulting shunting yard operations are feasible. As such, it arises at every shunting yard in the railway network and involves matching train units to arriving and departing train services as well as assigning the selected matchings to appropriate shunting yard tracks. We present a comparison benchmark of multiple solution approaches for this problem. In particular, we have developed a Constraint Programming formulation, a Column Generation approach, and a randomized greedy heuristic. We compare and benchmark these approaches against slightly adjusted existing methods based on a a Mixed Integer Linear Program, and a Two-Stage heuristic. The benchmark contains multiple real-life instances provided by the Danish State Railways (DSB) and Netherlands Railways (NS). The results highlight the strengths and weaknesses of the considered approaches.

6.1 Introduction

Passenger railway operation is an important mode of transportation in many countries. Commuters depend on a safe, reliable and timely operation that requires careful planning of trains, personnel and infrastructure. Planning the operations of a railway operator includes, among other things, *i)* determining a timetable, stipulating arrival and departure times of the train services to be operated, *ii)* creating a rolling stock schedule, specifying a feasible fleet circulation and

[1]Submitted to *Transportation Science*, October 2015

iii) constructing a crew plan, assigning train personnel to operate the trains. Due to the complexity of each the underlying optimization problems, the planning problems are usually resolved sequentially and in isolation.

A rolling stock schedule indicates which individual train units are allocated to each of the timetabled train services. If multiple units are assigned to the same train service, this results in a given train *composition*. Train services can, and do, have different compositions since the allocation of train units to services is done in such a way that the passenger demand is matched as closely as possible. The aim is typically to meet the passenger demand forecast, while not using more train units than is necessary. Whenever the composition changes between two successive train services, either a train unit is taken out of service or a train unit is brought into service. In the first case, the uncoupled unit must be *shunted* to the station's *depot* where it awaits its next service, while in the second case a unit must be retrieved from the depot and coupled to the train service. In both cases *shunting movements* are induced.

Shunting movements are often not considered when planning the rolling stock schedule. It is usually assumed that these can be resolved in a post-processing phase. The assumption is that the capacity as well as the infrastructure layout of any depot (i.e. *shunting yard*) is sufficient to cater for the induced shunting movements. In this paper we challenge this basic assumption. For railway networks where depot capacity is scarce, such as the suburban railway network in Copenhagen, it is not always possible to perform the shunting movements induced by a given rolling stock schedule. Furthermore, planning such movements is not a trivial problem. This is especially true at larger stations that have many shunting movements occurring over the course of a day and many depot tracks of different lengths on which to park train units not in service. Effective methods for finding feasible depot plans are essential.

In this paper, we present a comparison of optimization methods for determining whether the scheduled shunting movements at a given depot are feasible with respect to the number of depot tracks available. We term this problem the Train Unit Shunting Problem (TUSP) and benchmark several previously proposed methods from the literature on both realistic and artificial problem instances. In addition, we also test and compare three novel approaches for solving this problem. The first is a constraint programming approach, the second is column generation based approach, and the third is a greedy randomized heuristic approach. We view the TUSP as a feasibility problem only, as it ultimately determines whether or not a given rolling stock schedule is feasible. The majority of the operational cost is incurred in the rolling schedule and this would almost never be changed in order to improve the combined objective of the TUSP problems at each of the depots. It is important to simply know whether the induced shunting movements are feasible with respect to the depots. We note that, after confirming feasibility of the shunting movements, the TUSPs can be resolved with an appropriate objective function.

The set of shunting movements at each of the depots in the railway network can be deduced from a given rolling stock schedule. Scheduling the rolling stock is, however, beyond the scope of this paper; it is assumed to be given as input. From a rolling stock schedule all arrivals and departures from depots are implicitly specified. Railway operators typically have fleets of train units of different types. Two different unit types typically differ in their respective physical characteristics, e.g. length and passenger capacity. We assume rolling stock units of the same type to be interchangeable. The depots of two different stations are also assumed to be independent of each other. All shunting movements at a given depot are confined to that depot and have no impact on the shunting movements of other depots. A feasible solution to the TUSP, termed a *shunting plan*, must satisfy the following two constraints: the total length (or capacity) of each individual depot track is not violated at any given time, and no unit ordering *conflicts* are present. A conflict occurs when the arrival of a unit at a depot track blocks the departure of a unit from the same track. Depot tracks are assumed to function as last-in first-out (LIFO) queues, meaning the last unit to arrive at a depot track must be the first to leave. In reality, open ended depot tracks are allowed; however, such cases will also have ordering restrictions that must be obeyed. In this paper, any open-ended track is operated as a LIFO queue.

The TUSP entails finding a feasible shunting plan. However, in the absence of any possible solution, proving that no solution exists is equally as important. In such situations, a new rolling stock schedule must be found. The emphasis in this paper is on determining whether a solution exists. As mentioned above, no cost-structure is used to distinguish or rank distinct feasible solutions to a given instance of the TUSP.

We compare previously proposed methods and devise new methods to tackle the problem using different optimization techniques. These include: Constraint Programming, problem decomposition, Column Generation, and Mixed Integer Linear Programming with delayed constraint generation. Some of the proposed approaches are exact solution methods, which find one feasible solution (if it exists) or prove that no solution exists. The other methods are heuristic approaches that strive to quickly find any feasible solution by searching subsets of the entire solution space and consequently cannot prove infeasibility.

This research makes several contributions to the existing literature in this field. These include the following. First, we describe several efficient initial infeasibility checks for the TUSP. Secondly, we develop three novel methods for the TUSP: *i)* A constraint program formulation, *ii)* a column generation approach, and *iii)* a randomized greedy construction heuristic. We compare the performance of these against Mixed Integer Program (MIP) approach and a two-stage decomposition approach; two methods that have been proposed in the literature, for reference. Due to the high memory consumption of the MIP approach, we investigate the potential of delayed constraint generation for this method. A *track and unit type* decomposition approach is also presented that reduces a TUSP instance into several smaller and independent TUSP sub-problems. Furthermore, we discuss and present rolling stock platform parking, which is an interesting extension used by several railway operators in practice in order to circumvent capacity issues. Finally, all methods are benchmarked on TUSP instances from multiple railway operators residing in different countries. A few additional realistic benchmarks are performed in order to test different aspects of the presented methods.

The remainder of this paper is structured as follows. First, in Section 6.2 we give an overview of related literature and highlight the main differences to our contributions. In Section 6.3 we present the problem description. A number of polynomial solvable feasibility checks are discussed in Section 6.4 before introducing the solution methods in Section 6.5. The problem instances are presented in Section 6.6 followed by the benchmark of the solution methods. Finally, we conclude and give some remarks on further research in Section 6.7.

6.2 Literature Overview

To our knowledge, the TUSP was first introduced by Freling et al. [2]. Other authors have considered different variants of the same problem, including additional constraints and decisions such as maintenance operations or station routing. In some cases the train matching is given as input and not part of the problem. With the exception of Kroon et al. [8], all studies do not integrate the matching and parking problem, but solve them separately. A cost structure is often used to rank different matching and parking assignments. We, however, focus on the core matching and parking problem and do not differentiate between distinct solutions.

Freling et al. [2] consider the problem of parking train units overnight at a shunting yard in such a way that each unit can be retrieved when needed during the operations of the following day. Any feasible parking must ensure that train units of different types do not block each other when departing the yard the next day. The problem is decomposed into two smaller subproblems: a matching problem and a parking (or track allocation) problem. The first entails matching arrival and departure services at the shunting yard under consideration. A solution to this problem hence also stipulates the train services each physical unit will perform. This problem is formulated as a MIP and solved using a commercial solver. The parking problem, on the other hand, determines

how to park the assigned matchings on each of the tracks at the yard such that the track capacity is never exceeded and such that the movements associated with the matchings are conflict-free. The authors model this as a set partitioning problem with side constraints and solve it using a heuristic column generation procedure. A corresponding MIP approach was not considered nor compared. The proposed approach was tested on one instance from Netherlands Railways and results were found within 20 to 60 minutes of computation time. In contrast to our work, the authors adopt a cost-structure to both problems in order to rank the solutions. We compare our methods with a variant of this approach in Section 6.6.

The work of Lentink et al. [10] extends the work of Freling et al. [2], where a four-step approach is proposed to solve the matching and parking problem. The problems are still solved independently; however, the parking problem is extended to include more practical aspects, e.g. cleaning and maintenance of units, as well as the routing costs incurred from the shunting yard to the station platforms. The small problem instances are solved quickly, but the larger instances require at least 700 seconds of computation time.

A dynamic programming based heuristic approach for the TUSP is proposed by Haijema et al. [5]. The matching and parking problems are solved sequentially and in isolation. To reduce the problem size the authors propose a rolling horizon technique to solve the problem. A realistic test case from the railway station Zwolle in the Netherlands is used to analyse the performance of the algorithm. A 24 hour period is considered in which 45 units arrive and 55 units depart. The shunting yard has 19 tracks and a total capacity of 4000 metres. Solutions to the problems are found quickly and the results are promising; however, only a single instance is considered.

In contrast to Freling et al. [2], Lentink et al. [10], and Haijema et al. [5], we propose methods that integrate the matching and parking problems. In addition, we present and discuss the possibility of parking rolling stock units at platforms at the end of the planning horizon. Different models of the problem are presented and compared considering multiple different problem instances. Finally, as has been mentioned, we consider the TUSP to be a feasibility problem and hence do not differentiate between different, feasible plans.

Solving the TUSP without some form of matching/parking separation has been proposed by Kroon et al. [8]. The authors essentially extend the work of Freling et al. [2] and propose a large MIP formulation that simultaneously solves the matching and parking problems. The authors propose to minimize the number of splitted compositions, and in addition try to keep the depot tracks as homogeneous (w.r.t. unit type) as possible. In addition, they also consider conflict-cliques in order to reduce the large number of conflict constraints. In contrast, in our work we accommodate this problem by adding conflict constraints on the fly in the Branch-and-Bound (B&B) framework. Practical restrictions that include how to handle depot tracks that can be approached from both sides are described. Two stations from the Dutch Railway network form the computational study, where instances with up to 125 train units of 12 different types are considered.

Jacobsen and Pisinger [6] present three different heuristics to solve a variant of the TUSP that includes maintenance scheduling. The model is tested on small instances and the runtimes are low. Internal rearrangements are permitted if one unit is blocking another unit. The model is not tested on problem instances from practice.

6.3 Problem Description

In this section we give a formal problem description for the TUSP and introduce some general notation that is used throughout the paper. The core problem is to create a feasible shunting plan or prove that no such plan exists. A feasible shunting plan matches all initially parked units at the depot, as well as any units that arrive during the day, with compatible departures. Any unmatched unit must remain parked at the depot. Unmatched units reside in inventory until

Event	Type	Time
Arrival	a_1	12:00
Arrival	a_2	12:30
Arrival	b_1	13:00
Arrival	c	13:30
Arrival	b_2	14:00
Departure	b	15:00
Departure	c	15:30
Departure	a	16:00

Table 6.1: Example list of events in a problem instance. Note, that a_1 and a_2 (and b_1 and b_2) denote the same type, only different physical units.

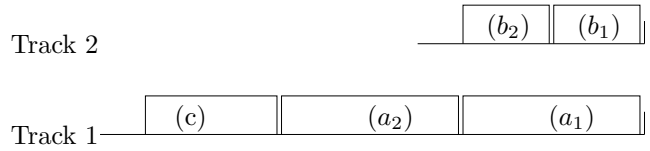


Figure 6.1: An example of units in Table 6.1 assigned to two depot tracks.

the end of the planning period. In a feasible matching all departures must be covered, otherwise the TUSP is infeasible. Given a feasible matching, the unit assignments to depot tracks must be conflict-free.

We term all units initially parked at the depot and all units that arrive during the day as *arrival* events. A specific unit type and known arrival time are associated with each arrival. The arrival time of initially parked units is assumed to be the start of the planning horizon t_0. Likewise, departing units are termed *departure* events. A departure has a known departure time and a required unit type. For the sake of simplicity, a departure is defined for all units remaining in the depot at the end of the planning horizon, t_∞. Consequently, a feasible matching covers all arrival events and all departure events. In other words, initially parked and arriving units are matched to either a compatible departure or assigned to stay on some track in the depot at time t_∞.

The matching and parking assignment must satisfy two types of constraints. First, the capacity on each individual depot track may not be violated at any time. In other words, the total length of all train units residing on a track at any time may not exceed the length of the track itself. It suffices to ensure that this holds whenever a train unit arrives at the depot. Second, all tracks must be processed in a LIFO order, i.e the last parked unit on a track must be the unit that leaves first. A unit cannot leave a track (for a departure) if a different unit has arrived in the meantime and is staying on the same track. We assume that an assigned track is occupied from (and including) the arrival time until (and including) the departure time of the corresponding matching. This is a conservative approach as an arrival will occupy its assigned depot track some time after arriving, and a departure will release the track allocation some time before departing from the associated station.

An example of a problem instance is given in Table 6.1, where a list arriving and departing events is specified. Each event occurs at a specific time and specifies the type of the units that is arriving, or the type that needs to depart. The lengths of the unit types are: $a : 200$, $b : 100$ and $c : 150$. The arrivals have to be matched to a departure or a stay in the depot (denoted by *inv*). Two matching examples are given in Table 6.2. Assume that we have two depot tracks available: track 1 of length 550 and track 2 of length 200. There is only one feasible parking possible using the track 1 and 2. This parking is shown in Figure 6.1. The matching to the left in Table 6.2 (Matching 1) is infeasible, because unit b_2 is blocking the departure of unit b_1. The other matching (Matching 2) is, however, feasible as no units are blocking any departure.

Matching 1	Matching 2
(b_1, b)	(b_2, b)
(c,c)	(c,c)
(a_1, a)	(a_1, a)
(a_2, inv)	(a_2, inv)
(b_2, inv)	(b_1, inv)

Table 6.2: Two examples of possible matching assignments for the problem instance in Figure 6.1.

The shunting movements at different stations in the railway network are completely independent of each other. Hence, the respective TUSPs can be solved separately for every station. No maintenance operations are considered in our problem. Thus, train units of the same type are completely interchangeable. Internal shunting of train units is also not considered; once a train unit is parked it can only be moved when retrieved for departure. In practice, internal shunting can be performed; albeit, at the cost of using shunting personnel resources. A plan without additional movements is preferred, if possible. Finally, a certain buffer period between consecutive arrivals and departures may be desired for the same train unit. We assume that a unit must be parked at a track for at least β minutes. The value of β is parametric. A high value of β will result in a shunting plan that is more robust to delays; however, it also reduces the combinatorial solution space. In the computational experiments we set β equal to one minute.

Most of the considered problem instances have an initial inventory of units at the start of the planning horizon. Each depot track may therefore contain a number of units in a specific order. The proposed approaches adhere to this initial ordering of the units. It is also possible for the presented approaches to determine the initial parking order of the units on the track. Not having an initial ordering is a less restrictive problem, potentially giving a greater number of feasible solutions. This variant will not be pursued further in this paper, but we note that it may be relevant in a strategic or tactical planning phase.

In contrast to other work in the literature, no cost structure is defined. The TUSP is considered a feasibility problem as this is the most prominent question to answer. The presented solution approaches can be used as part of a larger framework to determine whether or not a feasible solution exists before finding the most preferred one. This is highly applicable in an operational setting where time is limited and it is crucial to detect feasibility quickly. We note, however, that almost all of the presented approaches can, without much difficulty, be extended to include an objective.

Platform Parking

Rolling stock units can be, and are in certain situations, parked on passenger platform tracks in practice. During the day passengers board and alight from trains at platforms. Consequently, units should not be parked there during the day. However, during the night there is no, or limited, traffic. If units parked on platform tracks overnight can service the first train service of the following day, then no additional shunting is required.

According to some railway operators, e.g. Nederlandse Spoorwegen (NS) and Danish State Railways (DSB), trains are, under certain circumstances, parked on platforms. The rolling stock activities for the following day are known, and it can be practical to park train units on a platform overnight if the parked composition is to depart early the next day. Depending on the track layout and the number of platforms at the station, a number of platform tracks may be eligible to be used in this way. A number of platforms must, however, be reserved in order to allow night trains or maintenance crews to operate.

At any station we assume that a certain number of platform tracks, N, can be used for overnight

Set	Definition
E	Set of all events
$E^{arr} \subset E$	Set of arrival events
$E^{dep} \subset E$	Set of departing events
M	Set of train unit types
S	Set of tracks
$S_d \subset S$	Set of tracks in the depot for parking
$S_p \subset S$	Set of platform tracks

Table 6.3: List of defined sets

Notation	Description
t_e	Time event $e \in E$ takes place
m_e	Rolling stock type corresponding to event $e \in E$
l_m	Length of rolling stock type $m \in M$
c_s	Length of track $s \in S$

Table 6.4: List of defined shorthand notations

parking. In order to ensure a smooth operation, the first N departing train services (of the the following day) dictate which units can be parked on the N available platform tracks. A departing train service may consist of multiple units, allowing more than a single unit to be parked on a platform track. For instance, if $N = 2$, and the first two departing trains have the compositions aa and bc, then two units of type a can be assigned to the first platform (even coming from different arrival services), and a single unit of each type b and c on the second platform. In the considered variation of the problem, the extension can be handled by adding the N platform tracks as normal shunting tracks with additional restrictions. More specifically, in addition to the existing track constraints, an arrival and departure matching can only be assigned to the platform track if the arrival takes place in an appropriate time window (close to the end of the daily operation) and if the departure corresponds to one of the first train services of the following day.

6.3.1 General Notation

In the TUSP, a set of incoming and outgoing events occurring at a shunting yard are assumed to be given. This set of events is denoted by E. Each event $e \in E$ is either an arrival or a departure. The sets E^{arr} and E^{dep} respectively contain all arrivals and departures, and together define a partition of all events; that is, $E^{arr} \cup E^{dep} = E \wedge E^{arr} \cap E^{dep} = \emptyset$. An arrival corresponds to a unit that is uncoupled at the station and must be parked, while a departure is a unit that is to be coupled at the station and must be retrieved from the shunting yard. Furthermore, we assume a set, M, of the rolling stock types and that a set, S, of shunting yard tracks are available in the depot. Two subsets of $S_d \cup S_p = S$ denote the set of depot tracks and the set of platform tracks. All definitions are summarized in Table 6.3.

The time at which event $e \in E$ takes place is denoted by t_e, and m_e denotes the train unit type of the corresponding event. We let c_s denote the parking capacity of track $s \in S$. This capacity is equal to the maximum length that can be stored simultaneously on track $s \in S$. The length of a train unit type $m \in M$ is assumed to be l_m. Table 6.4 summarises this notation.

Complexity

The complexity of matching and parking trains has been addressed by multiple authors in the literature. Multiple variants of the problem exist, which are known to be $\mathcal{NP}$-hard. The shunting

problem considered by Freling et al. [2] is essentially a specialization of the considered TUSP. They prove the variant to be $\mathcal{NP}$-hard by reduction from the Tram Dispatching Problem studied by Winter et al. [14], which in turn is $\mathcal{NP}$-hard.

We present a simple and informal proof, that shows that the considered TUSP is $\mathcal{NP}$-hard by reduction from the Graph Coloring Problem (GCP). In the GCP a color must be assigned to each vertex such that no adjacent vertices have the same color. Two vertices are adjacent if an edge connects them. The problem is then to decide the fewest number of colors needed. The corresponding decision problem is to decide whether a graph can be colored using k colors. The GCP is known to be $\mathcal{NP}$-hard [3].

First, the TUSP is in $\mathcal{NP}$ since a feasible solution can be verified in polynomial time. The matching constraints are verified by counting the number of assignments, the capacity constrains can be verified by looping through the events (ordered by time), and a pair-wise comparison of all matching assignments to tracks can verify that no ordering conflicts exist. Second, we argue that the TUSP is a generalization of the GCP. Given a instance of the GCP, the number of colors corresponds to the number of available tracks. The length of the tracks is set sufficiently high, such that it is never binding. The constructed TUSP instance is generated such that only one valid matching exists by assigning a unique unit type to all train units. A vertex corresponds to a track assignment (of a matching), and the selected track represents the selected color. The edges of the graph are now used to generate a relative ordering of the arrival and departure events such that two matchings are in conflict if and only if an edge exists between the corresponding vertices in the graph. If the constructed TUSP instances contains a feasible solution, then a feasible assignment of k colors is given by the track assignments of the matchings of the TUSP instance.

6.4 Infeasibility Checks

A select number of efficient feasibility checks can be performed independently of any solution method. The problem instances considered in the benchmark testing of Section 6.6 have all passed the checks discussed in this section. There is no reason to consider an instance, which violates any of the following checks as it is inherently infeasible.

Aggregated Track Capacity

At any given time the sum of all depot track lengths must be no less than the sum of all train units that need to be parked. This aggregated constraint must hold since no feasible solution can exist if it is violated. This property is easily checked in polynomial time. Rolling stock schedules implicitly satisfy this constraint if depot capacity is modeled in the rolling stock problem.

Individual Track Capacity

The depot must have at least one feasible initial parking. At the start of the planning horizon a feasible parking must exist, otherwise no solution can exist to the TUSP either. With a given set of initial units, a feasible solution can be found by solving a 0-1 Multiple Knapsack Problem (MKP). Every train unit corresponds to an item, where the capacity consumption is equal to the physical length of the unit. Each depot track corresponds to a knapsack, where the capacity is equal to the track length. Efficient algorithms for solving the MKP exist (see e.g. Pisinger [13]), making this check efficient; especially since the resulting MKP problem size is small.

In this paper, we assume that the initial parking given is feasible with respect to the mentioned knapsack constraints. However, the same constraints must be satisfied during the whole planning period, and not only for the initial parking. The same check is applied every time a unit arrives

Set	Definition
$\mathcal{A}$	Set of all matchings
$\mathcal{A}_e^{arr}$	Set of matchings where event $e \in E^{arr}$ is the arrival
$\mathcal{A}_e^{dep}$	Set of matchings where event $e \in E^{dep}$ is the departure

Table 6.5: List of MIP specific sets

to the depot. Note that the individual track capacity can be violated even if the aggregated track capacity is satisfied.

Feasible Matching

In the common case, multiple feasible matching exists for the same problem instance, especially when ignoring depot track capacities. The reason for integrating matching and parking in the TUSP is the fact that all feasible matchings do not necessarily have a feasible parking assignment. However, if no feasible matching exists then the TUSP is infeasible as well. Detecting whether a feasible matching exists is equivalent to solving the Assignment Problem (AP) (Munkres [12]), which is solvable in polynomial time as the resulting Linear Program (LP) is totally unimodular. As the number of arrival and departure events are equal in size, the problem is the linear AP; this can be solved using specialized polynomial time algorithms, such as the Hungarian algorithm (Kuhn [9]).

6.5 Solution Methods

In this Section we describe the different solution methods for solving the TUSP. First, the Reference MIP Method (RMM) is described in Section 6.5.1. In Section 6.5.2 the Constraint Programming Method (CPM) is presented. A column generation method and the Two-Stage Method (TSM) are presented in Section 6.5.3. Finally in Section 6.5.5 we introduce the Randomized Greedy Construction Heuristic (RGCH).

6.5.1 The Reference MIP Method

Arrivals and departures are linked using matchings. The matching of an arrival and a departure event is allowed if and only if sufficient time separates the events and the unit types are compatible. The set of all possible matchings is denoted by $\mathcal{A}$:

$$\mathcal{A} = \{ \quad (e,f) \quad | \quad t_e + \beta \leq t_f, \quad m_e = m_f, \quad e \in E^{arr}, \quad f \in E^{dep} \quad \}$$

The set of matchings where event $e_1 \in E^{arr}$ is the arrival is denoted by $\mathcal{A}_{e_1}^{arr}$ and the set of matchings where event $e_2 \in E^{dep}$ is the departure is denoted by $\mathcal{A}_{e_2}^{dep}$. These additional sets are summarized in Table 6.5.

The mathematical model contains one family of binary decision variables $X_{a,s}$. The variable $X_{a,s}$ takes a value of 1 if and only if matching $a \in \mathcal{A}$ is selected and parked on track $s \in S_d$. We extend the model to include platform parking later.

A number of constraints need to be satisfied in order to achieve a feasible solution. First, each arrival must be assigned to exactly one departure, c.f. Constraints (6.1). Similarly, each departure

must be assigned to exactly one arrival, c.f. Constraints (6.2).

$$\sum_{s \in S_d} \sum_{a \in \mathcal{A}_e^{arr}} X_{a,s} = 1 \qquad \forall e \in E^{arr} \tag{6.1}$$

$$\sum_{s \in S_d} \sum_{a \in \mathcal{A}_f^{dep}} X_{a,s} = 1 \qquad \forall f \in E^{dep} \tag{6.2}$$

Constraints (6.1)-(6.2) ensure a feasible matching. The capacity of a depot track can not be exceeded at any point in time. It is therefore sufficient to ensure that the track capacity is not exceeded at every arrival, c.f. (6.3).

$$\begin{aligned} &\sum_{\{e' \in E^{arr} | t_{e'} \leq t_e\}} \sum_{a \in \mathcal{A}_{e'}^{arr}} X_{a,s} \cdot l_{m_{e'}} \\ &- \sum_{\{e' \in E^{dep} | t_{e'} \leq t_e\}} \sum_{a \in \mathcal{A}_{e'}^{dep}} X_{a,s} \cdot l_{m_{e'}} \leq c_s \qquad \forall e \in E^{arr}, s \in S_d \end{aligned} \tag{6.3}$$

For each arrival, Constraints (6.3) sum the contribution of past events and ensure that the used track length is less than the available track length; arrivals consume capacity, while departures release capacity.

The depot tracks are subject to LIFO restrictions. Only the out-most (top of stack) can be retrieved at any point in time. We model these restrictions by adding one constraint per pair-wise conflict to disallow such assignments, c.f. Constraints (6.4). It states that any two pairs of matchings cannot be assigned simultaneously if they block each others' movements.

$$X_{a,s} + X_{a',s} \leq 1 \qquad \forall s \in S_d, (a, a') \in \mathcal{C} \tag{6.4}$$

where

$$\begin{aligned} \mathcal{C} = \{ \quad (a, a') \quad | \quad & a = (e, f) \in \mathcal{A}, \\ & a' = (e', f') \in \mathcal{A}, \\ & t_{e'} < t_e \ \wedge \ t_{f'} < t_f \ \wedge \ t_{f'} > t_e \quad \} \end{aligned}$$

Note, that the conflict set is not track-dependent. All pair-wise conflicts are therefore repeated for every track, effectively generating very many constraints. We note that the number of constraints in the model can be reduced, possibly drastically, by replacing the pair-wise conflicts with conflict cliques, see Kroon et al [8]. In general, the problem of finding such cliques is $\mathcal{NP}$-hard (Karp [7]). However, a number of cliques can be found heuristically in order to make the problem more tractable.

The initial inventory is not modeled directly in the above formulation. Recall, that initially parked units are modeled using arrivals and that parking (at the end of the planning horizon) is modeled using departures. Any initial unit to track assignment can be modeled by fixing the variables corresponding to the first set of events. Any order of units on depot track can also be achieved by shifting the artificial arrival times (without exceeding the real first event) to reflect the same ordering. Likewise, any final parking can be achieved using a similar modification for the artificial departures.

In the computational experiments of Section 6.6, a parking order is imposed. We note that in the implementation of the model all fixed variables are removed in order to obtain a more compact model.

Set	Definition
C	Set of possible compositions
Q	Set of possible composition changes
$Q_{e,s}$	Set of composition changes that are allowed after event $e \in E$ at track $s \in S$

Table 6.6: List of all additional sets required by the CP model

Due to the large number of (conflict) constraints present in the model, we also introduce the Delayed Constraint MIP Method (DCMM), where these constraints are generated on-the-fly. The DCMM is solved as a MIP model, where violated conflict constraint are added as they become violated by the optimal LP solutions. Initially, no conflict constraints are added. The success of this approach depends on the fact that most conflicts will never be violated in the B&B approach of a MIP solver.

Platform Parking Extension

A given set of platforms $s \in S_p$ is available for parking arrival trips overnight. The set of decision variables is extended to include platform parking. The model now contains one variable $X_{a,s}$ for every activity $a \in \mathcal{A}$ and track $s \in S = S_d \cup S_p$. We denote the number of slots at track $s \in S_p$ for units of type $m \in M$ to be $p_{m,s}$. Constraints (6.5) ensure that the number of parked units does not exceed the number of units that can be parked on the platform.

$$\sum_{a \in \mathcal{A}_e^{arr}} X_{a,s} \leq p_{m,s} \qquad \forall m \in M, e \in E^{last}, s \in S_p \tag{6.5}$$

No LIFO ordering contains are included as the units on the track leave as a whole, and the track capacity is implicitly satisfied by construction of the $p_{m,s}$ coefficients. Preferably, only arrivals close to the end of day can be matched to the platforms. This is enforced by fixing or removing the invalid matching variables.

6.5.2 The Constraint Programming Method

As it is primarily a feasibility problem, the TUSP can be formulated using a Constraint Program (CP) approach. Our proposed formulation is inspired by the rolling stock composition model of Fioole et al [1], where it was originally used for Rolling Stock (Re)Scheduling; however, we use the idea of compositions and composition changes for the TUSP. Instead of assigning events to tracks, we assign compositions to tracks. For every time-interval one composition is assigned to every track individually. A composition consists of a number of train units in a specific order. Note that the empty composition, containing no train units, is a valid composition.

We let C be the set of all possible compositions and Q be the set of all possible composition changes. The set $Q_{e,s}$ consists of all feasible composition changes that can take place just after event $e \in E$ on track $s \in S$. For instance, if event e stipulates that a unit of type a is arriving, then only composition changes where a unit of type a appears on the top of the stack are included in $Q_{e,s}$. Additionally, composition changes where no units are appended or removed are included as all unaffected tracks remain unchanged. See Table 6.6 for an overview of the additional sets.

The first additional parameter required by the CP is i_s, which specifies the initial composition on track $s \in S$. Next, λ_e defines the predecessor event of event $e \in E$, i.e. the event that occurs just before e. Furthermore, for composition change $q \in Q$, we introduce the shorthand notation,

Parameters	Description
i_s	The composition belonging to the start inventory at track $s \in S$
λ_e	The predecessor event of event $e \in E$
$\texttt{In}[q]$	The index of the first composition belonging to composition change $q \in Q$
$\texttt{Out}[q]$	The index of the second composition belonging to composition change $q \in Q$
$\alpha_m[q]$	Equals 1 if a unit of type $m \in M$ is appended to the composition on the track during composition change $q \in Q$
$\beta_m[q]$	Equals 1 if a unit of type $m \in M$ is removed from the composition on the track during composition change $q \in Q$

Table 6.7: List of all parameters

$\texttt{In}[q]$ and $\texttt{Out}[q]$, which denote the index of the first and second composition in a composition change. Thus, the original composition and its successor composition. Finally, $\alpha_m[q]$ and $\beta_m[q]$ specify whether a unit of type m is appended or removed. These parameters are summarized in Table 6.7.

We define two families of decision variables. First, the integer variable $X_{e,s}$ specifies which composition is assigned track $s \in S$ just after event $e \in E$. The compositions are mapped to integer values, e.g., $X_{e,s} = 3$ stipulates that the *ab* composition is assigned to track s after event e. Recall, a composition $c \in C$ which is assigned to track $s \in S$ just after event $e \in E$ consists of all train units parked at that moment on track s, in order of arrival time. For instance, if the composition *abcd* is assigned track s after event e, it means that unit d was parked there first, thereafter unit c, then b, and finally train unit a.

The integer decision variable $Y_{e,s}$ represents the composition change that is performed on track $s \in S$ just after event $e \in E$. An example is the composition change from composition *aa* to *a*, where one unit is removed. Again, the integer values are mapped to the change from a specific composition to another specific composition.

The construction of the $Q_{e,s}$ set models the allowed compositions changes with respect to the depot track capacity and the LIFO restriction. Note that platform parking can also be modeled by construction of this set. First, composition changes that exceed the length of a track are not allowed. All composition changes that involve a transition to a composition that has a total length longer than the capacity of track s are removed from the set $Q_{e,s}$. Furthermore, restrictions with respect to the unit type of events are considered. First, if event $e \in E$ is an arrival of a unit of type a, then only composition changes where a unit of type a is appended and composition changes where no units are appended or removed are allowed. Second, the LIFO constraints further restrict the set of composition changes. If, for instance, the composition *abcd* was assigned to track s just after event λ_e, then there are only 2 allowed composition changes after a departure $e \in E$: $abcd \rightarrow abcd$, or $abcd \rightarrow bcd$. Units b, c, and d cannot depart as unit a is blocking them. Finally, if platform parking is allowed, as it is not allowed to park train units at platforms during the day, the only allowed composition change after events during the day is $empty \rightarrow empty$. For the last events of the day, however, platform parking can be considered. In such cases the platform track composition is restricted to being a subset of the composition of the train service which will first depart from the platform on the following day. This can be considered by only allowing those composition changes that involve transitions to compositions which are a subset of the specific departing train service composition for the following day.

The TUSP can be modeled with the following mathematical program, which is used as a basis for the CPM:

$$X_{\lambda_e,s} = \texttt{In}[Y_{e,s}] \qquad \forall e \in E, s \in S \tag{6.6}$$

$$X_{e,s} = \texttt{Out}[Y_{e,s}] \qquad \forall e \in E, s \in S \tag{6.7}$$

$$\sum_{s \in S} \beta_m[Y_{e,s}] = 1 \qquad \forall e \in E^{dep}, m \in M : m_e = m \tag{6.8}$$

$$\sum_{s \in S} \alpha_m[Y_{e,s}] = 1 \qquad \forall e \in E^{arr}, m \in M : m_e = m \tag{6.9}$$

Constraints (6.6) state that the first composition of a chosen composition change on a track has to match the actual composition that is appointed before the composition change took place on the track. This actual composition is the composition that is assigned to the track just after the previous event, λ_e. A similar composition flow conservation constraint is used for the second composition of a chosen composition change. This composition must be equal to the actual composition that is appointed to track after the composition change took place, which equals the composition that is assigned to the track just after the event, e. This is modeled by Constraints (6.7).

Every departure event has a corresponding unit that has to depart from precisely one of the tracks in the depot. Consequently, exactly one composition change has to be selected where a unit of type m_e is removed; on all other tracks no shunting movements can take place. This is modeled by Constraints (6.8). A similar requirement holds for an arrival. The corresponding unit m_e has to be appended to precisely one track; this is handled by Constraints (6.9).

Finally, the start inventory is enforced by fixing the values for $X_{e,s}$ of an auxiliary source event e that occurs before the first event.

Solution procedure

The proposed model can be solved using a CP solver, effectively solving the TUSP by assigning compositions and composition changes to the events on the tracks. Similar to the rolling stock scheduling variant in [2], the model does not scale well. Long depot tracks and a high number of unit types result in a huge number of variables in practice. A large number of variables is needed for long depot tracks as the number of possible compositions increases drastically in such cases. Multiple, different unit types further increase the combinatorial solution space.

Preliminary results showed that small instances with two unit types are practical to solve. However, larger instances with four unit types quickly become impractical to solve primarily due to the memory footprint. As a remedy, the CPM can be adapted to a more practical heuristic method. We present a heuristic variant, the Constraint Programming Heuristic (CPMH), that restricts the compositions on tracks to contain at most $\epsilon \geq 1$ different unit types. Note, that the contained types can change over the course of a planning horizon. At any time, the number of different types is at most ϵ. In effect, fewer tracks will be mixed, i.e., contain more than one unit type, in the solution. Keeping tracks homogeneous is beneficial as it also reduces the potential LIFO conflicts. The main benefit is the drastic reduction of variables in the CP model. If the heuristic finds a solution, then clearly it is feasible for the TUSP. However, if it fails to find a solution we cannot conclude that the solution is infeasible, unless ϵ is equal to the actual number of unit types.

The minimal value of ϵ that produces a feasible solution is initially unknown, therefore we begin with $\epsilon = 1$. The strategy is to solve the model using increasing values of ϵ. A time limit of γ minutes is set in every iteration. Otherwise, too much time is potentially spent searching an infeasible solution space. In this paper we divide the available time, T, uniformly by the number of unit types in the problem instance, $\gamma = {}^{T}/_{|M|}$. In every step when no solution has been found, we increase ϵ with 1 and try again, until $\epsilon = |M|$.

6.5.3 The Column Generation Method

To combat the potentially large number of constraints in the MIP formulation, a column generation approach can be used to solve the TUSP. In this approach the problem is decomposed by track, where each track is assigned to a set of possible matchings, termed a *matching pattern*. A matching pattern is a subset of matchings that can be feasibly parked on a given track over the planning horizon. In particular, it is a set of matchings that satisfies the LIFO requirements as well as the available track length restriction. A large number of possible matching patterns exist thus the approach relies on dynamic generation of variables that represent promising matching patterns. In this paper, the solution method is referred to as the Column Generation Method (CGM).

The proposed formulation is based on the methodology presented by Freling et al. [2], with the exception that in our work the matching and parking problems are not solved separately. We present a model and solution framework that simultaneously solves both problems. To assist in the description of the model, we introduce the set $\mathcal{P}_s$, which denotes the set of all feasible matching patterns for track $s \in S$. Note, that platform parking and LIFO constraints are already satisfied in this set. A binary decision variable $X_{p,s}$ is defined for each $s \in S$ and $p \in \mathcal{P}_s$ and governs the inclusion of the corresponding matching pattern in the final solution. A value of one indicates that the matching pattern is chosen, while a value of zero indicates otherwise. As the majority of the constraints are embedded in the column construction phase, the problem can be formulated as a large generalized set partitioning problem. In the model, at most one matching pattern is assigned to each track. Further, each arrival and departure must appear in at most one matching pattern, otherwise it is left uncovered. Binary variables Y_e, where $e \in E^{arr}$, and Z_e, where $e \in E^{dep}$, are used to indicate whether an arrival, respectively departure, is matched or not. These variables are penalized in the objective function by, Γ, thus providing an incentive for the model to match as many events as possible. Finally, the binary parameter $\alpha_{e,p}$ indicates whether or not event $e \in E$ is contained in matching pattern $p \in \mathcal{P}_s$. The full binary integer program is given as follows.

$$\text{Minimize:} \quad \sum_{e \in E^{arr}} \Gamma Y_e + \sum_{e \in E^{dep}} \Gamma Z_e \tag{6.10}$$

subject to:

$$\sum_{p \in \mathcal{P}_s} X_{p,s} \leq 1 \qquad \forall s \in S, \ (\boldsymbol{\mu}) \tag{6.11}$$

$$\sum_{s \in S} \sum_{p \in \mathcal{P}_s} \alpha_{e,p} X_{p,s} + Y_e = 1 \qquad \forall e \in E^{arr}, \ (\boldsymbol{\pi}) \tag{6.12}$$

$$\sum_{s \in S} \sum_{a \in \mathcal{P}_s} \alpha_{e,p} X_{p,s} + Z_e = 1 \qquad \forall e \in E^{dep}, \ (\boldsymbol{\gamma}) \tag{6.13}$$

$$X_{p,s} \in \{0,1\} \qquad \forall s \in S, p \in \mathcal{P}_s, \tag{6.14}$$

$$Y_e \in \{0,1\} \qquad \forall e \in E^{arr}, \tag{6.15}$$

$$Z_e \in \{0,1\} \qquad \forall e \in E^{dep}. \tag{6.16}$$

The objective, given in (6.10), minimizes the penalties incurred from uncovered events. Constraints (6.11) ensure each depot track is assigned at most one matching pattern. Constraint sets (6.12) and (6.13) enforce the requirement that each arrival and departure appears in one of the selected matching patterns, or is left uncovered. Finally, variable domains are specified by constraints (6.14)-(6.16). We refer to Model (6.10)-(6.16) as the *master* problem.

The Master Problem

Given the exponential number of matching patterns in any real-life example it is impractical to enumerate all corresponding columns and solve this formulation. In our solution method, a subset (restricted set) of the possible matching patterns are included. We relax the integrality restrictions and associated bounds given by (6.14)-(6.16). A *relaxed, restricted master problem* (RRMP) is obtained. Using the optimal dual solution vector $(\boldsymbol{\mu}^*, \boldsymbol{\pi}^*, \boldsymbol{\gamma}^*)$ to this relaxed problem, a *pricing* problem is solved to determine if any favourable matching patterns exist. Promising variables are inserted iteratively into the restricted master problem until none exist - implying that the LP solution is proven optimal. By iterating between the RRMP and several pricing problems (typically one for each track), one can limit the search for the optimal solution to model (6.10)-(6.16) to include only those matching patterns that have the potential to improve the objective value. For a general introduction to column generation the reader is referred to [11].

The Pricing Problem

The pricing problem requires one to find a favourable set of matchings that can feasibly be parked on a given track. In other words, given an optimal solution to the RRMP, one must solve up to $|S_d| + |S_p|$ pricing problems at any column generation iteration to determine if any improving matching pattern exists. To find such patterns we present an approach that finds shortest paths in a directed *pricing* graph.

In the *pricing* graph there is one node for every possible matching, one node for every arrival (corresponding to not parking the arrival), and a source and sink node. The graph is layered by matchings for each arrival (including the node corresponding to not parking the arrival) and these layers are ordered in increasing arrival time. Arcs connect matchings in one layer with those of the subsequent layer - providing the two matchings can feasibly use the same track. The source node is connected to each matching in the first layer, while each matching in the last layer is connected to the sink. There is a cost on any arc entering a matching node equal to dual contribution to the reduced cost of the arrival and departure matched in the matching. E.g. if events $e \in E^{arr}$ and $e' \in E^{dep}$ are matched, the cost on any arc entering the node corresponding to this matching will have a cost of $-(\pi_e + \gamma_{e'})$. An example of such a network is given in Figure 6.2

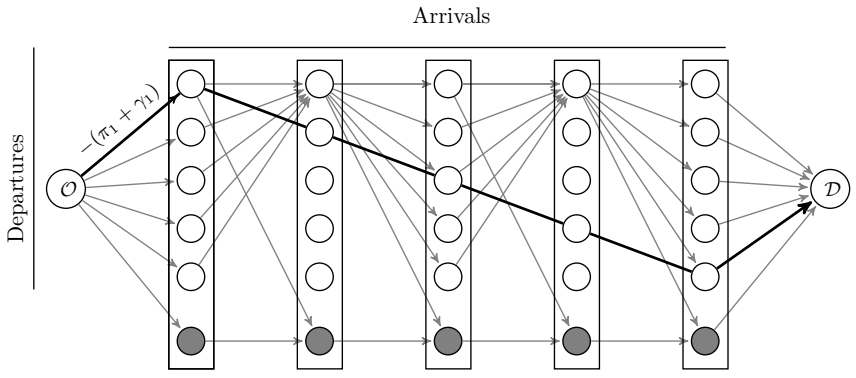


Figure 6.2: An example subproblem network with five arrivals and five departures. Every node corresponds to either the matching of an arrival and a departure or an unmatched arrival (gray nodes). All paths originating at the source ($\mathcal{O}$) and terminating at the sink ($\mathcal{D}$) represent a matching pattern, which is feasible if the resource constraints are respected. An example of such a path is given in black. Note that not all arcs and costs are shown.

As we must observe the available track length and satisfy the LIFO requirement when generating matching patterns, a resource constrained shortest path problem must be solved. Consequently,

a standard label setting algorithm is used to identify paths in this network. The algorithm is similar to that of Freling et al. [2]; however, as we must also simultaneously find the matching, the proposed network is much bigger. Additionally, we must also keep track of previously matched departures. The ordering of the layers ensures that each arrival is matched exactly once (or not parked); however, a path in the presented graph can match the same departure more than once. This is indeed likely if doing it improves the reduced cost resulting in matching patterns that cover the same departure multiple times. In addition to being infeasible, these patterns also dominate feasible labels. For an exact approach, the only option would be to weaken the dominance (ensuring all possibilities) are generated; however, this is impractical with large networks. In this paper, we adopt the second approach; i.e. a label can not visit a matching if the departure associated with the matching has been previously matched. This is heuristic; however, it ensures the solution times of the pricing problem are manageable.

To ensure exactness of the column generation, the heuristic column generation approach outlined can be complemented with a MIP solve. For instance, one can resort to a MIP when the column generation fails to identify a negative reduced column. This MIP, however, would be similar in structure to that described in Section 6.5.1, with the exception that only the "best" set of matchings need to be decided for the shunting track in question. For large problems, this is expected to be slow.

6.5.4 The Two-Stage Method

Given a matching to an instance of the TUSP, the remaining problem simply entails parking the set of matchings. A problem instance may contain multiple feasible matchings for which a feasible parking exists. Solving these two problems in isolation is expected to be easier than solving them jointly. This motivates the Two-Stage Method (TSM), where in the first stage a feasible matching of arriving and departing units is generated, while in the second stage the method tries to park the found matchings.

The matching and the parking problem can be solved using different methods. For the matching problem we resort to a MIP approach for two reasons. First, it is easy to formulate and implement a MIP for the matching problem. Second, as mentioned in Section 6.4, the resulting LP is totally unimodular. We also adopt a MIP approach for solving the parking problem. Freling et al. [2] proposed a column generation approach for solving this problem; however, the fast runtimes of our approach gave us no reason to pursue a more complicated framework.

For the matching problem formulation a binary decision variable X_a is introduced that indicates whether a given matching $a \in \mathcal{A}$ is selected or not. In a feasible matching, each arrival and departure should appear in exactly one matching. The resulting constraints of the MIP are given below.

$$\sum_{a \in \mathcal{A}_e^{arr}} X_a = 1 \qquad \forall e \in E^{arr}, \tag{6.17}$$

$$\sum_{a \in \mathcal{A}_e^{dep}} X_a = 1 \qquad \forall e \in E^{dep}, \tag{6.18}$$

$$X_a \in \{0,1\} \qquad \forall a \in \mathcal{A}. \tag{6.19}$$

There is no objective used in the MIP, since we are interested in feasibility only. Constraints (6.17) and (6.18) ensure, respectively, that each arrival and departure event appears in exactly one matching. The variable domains are given by (6.19). Several solutions, i.e. matchings, to the model may exist and it may therefore be useful to guide the solution in a more advanced approach.

If a solution to the matching problem problem exists, we proceed to the second stage and attempt to park them. For this, we use the MIP described by Haahr et al. [4], which is identical in structure to the reference MIP approach described earlier.

The TSM is similar to what is described by Freling et al. [2]. However, the matching problem we propose is slightly different and we use a MIP solver for the parking problem. A heuristic column generation framework is described in Freling et al. [2].

6.5.5 The Randomized Greedy Construction Heuristic

Modeling ordering constraints efficiently in integer linear programs such as LIFO constraints is cumbersome. Our proposed solution methods overcome this modelling issue by either adding all pairwise conflicts, enumerating all possible transition states, or by generating feasible parking patterns. All methods have scaling issues, by the number of constraints or variables. In contrast, modelling one or multiple stacks programmatically is fairly straightforward.

We propose a heuristic that greedily assigns arrivals and departures to and from tracks. The important key ingredients are the efficiency of the construction and the randomization of the greedy choice. Together these characteristics allow the heuristic method to try multiple paths of track assignments and extractions within a short time. The method terminates with the first feasible solution.

Algorithm 1 Randomize Greedy Construction Heuristic

```
1: Input: Track set S
2: Input: Event set E, ordered by time
3: Output: Matching set M
4: M ← ∅
5: S ← InitializeEmptyTrackStacks()
6: for e ∈ E do
7:     if Type(e) =Arrival then
8:         s ← FindRandomCompatibleTrack(S)
9:         if s = ∅ then
10:            return ∅
11:        else
12:            S[s] ←Push(S[s], e)
13:    else
14:        s ← FindRandomCompatibleUnitType(S)
15:        if s = ∅ then
16:            return ∅
17:        else
18:            (S[s], e′) ←Pop(S[s])
19:            M ← M ∪ {(e, e′, s)}
```

An overview of the heuristic is shown in Algorithm 1. The input to the heuristic is the set of events to process and the set of available tracks. The main loop processes events by ascending time. In case of an arrival, a random compatible track is sought, i.e., any track that has sufficient remaining length to hold the arriving train unit. Many candidates may exist, thus the following selection criteria are used:

1. A track where the existing outmost unit has the same type
2. A track which is empty
3. Any track with sufficient capacity

The goal is to group the same type of units, and avoid stacking different unit types on the same tracks. Note, units of the same type do not block each other as they are interchangeable. In order to avoid a standstill in certain situations at depots with scarce capacity, the first and second criteria are skipped with a low probability. The situation occurs when unit types must be mixed on tracks in order to utilize the capacity fully.

In case of a departure, tracks are processed in a random order. The track with the correct unit type (on the top of the stack) is selected. Here it might be worthwhile considering a selection criteria approach based on the effect of removing this unit. However, preliminary results show that this simple extraction rule is sufficiently effective.

The algorithm output is a list, where every element defines an arrival, a departure matching, and a specific track. On arrival the unit is parked on the track, and the specified departure extracts the unit from the same track.

The heuristic is able to evaluate one construction path very quickly, thus it is embedded in an iterative loop, where the heuristic is applied with different seeds to initialize the random number generator. Every iteration thus essentially restarts the whole process. The loop continues until either a feasible solution is found or the time limit is reached.

6.5.6 Type and Track Decomposition

Some problem instances contain many events, unit types, or long tracks. This results in a large number of possible matchings or track assignments, which makes the problem impractical to solve using exact methods. For example, the CPM and RMM require too much memory to represent the mathematical formulas as they contain an explicit representation of the problem.

The solution space can be reduced significantly by decomposing the problem instances by unit types and tracks. In the proposed decomposition, a unit type is restricted to park on a select subset of tracks. The partitioning of the tracks and unit types can be performed such that the original problem decomposes into several independent problems, which can be solved individually in sequence or in parallel. We consider such partitions where both the unit types and tracks are partitioned into K groups, such that one group of unit types is assigned to one group of tracks. By construction, no interaction needs to takes place across the selected groups.

The decomposition divides the problem into a number of smaller independent subproblems. The primary advantage is that solving all resulting subproblems is easier than solving the full original problem. A second advantage is that the decomposition is independent of the underlying solver. The subproblems can be solved using any solution method for the TUSP. The primary disadvantage is that the resulting framework is inherently heuristic as the decomposition restricts the original solution space. Feasible solutions found using the decompositions are naturally also feasible in the original problem; however, we cannot conclude that a problem instance is infeasible if any one subproblem is infeasible. Another drawback of the proposed decomposition is the existence of multiple partitions. Some of the partitions may contain feasible solutions to all subproblems while others may not. Determining the partition is therefore another problem that must be addressed.

Due to the scope of this paper, we only propose a simple method of finding eligible partitions that will be tested in Section 6.6. For the selected problem instances we generated a number of random partitions. Partitions are rejected if they do not pass the checks described in Section 6.4.

Class	Depot	Min	Max	Tracks	Length	Types
STOG	BA	12	14	4	936	2
STOG	FM	16	66	4	727	2
STOG	FS	26	58	6	1 020	2
STOG	HI	20	54	6	1 635	2
STOG	HOT	2	22	1	173	2
STOG	HTAA	20	68	35	3 272	2
STOG	KH	36	78	9	2 753	2
STOG	KJ	24	60	6	1 115	2
STOG	KL	6	66	3	558	2
STOG	UND	24	46	6	1 670	2
DSB	KK	326	518	10	3 878	12

Table 6.8: Summary of instances: The first column indicates to which class the problem belongs. The second column defines the station. The third and fourth column show the minimum and maximum number of events taking place at the station. The fifth column presents the number of depot tracks available within the station and the sixth column defines the total length of all depot tracks combined. Finally, the seventh column defines the number of different rolling stock types that need to be parked within the station.

6.6 Computational Results

The presented solution methods are benchmarked on different classes of instances, which originate from three different railway networks in two different countries. Four classes, summarized in Tables 6.8, 6.9 and 6.10, are considered: STOG, DSB, NS, and NS-HARD. These instances are based on the railway networks of the Danish State Railways (DSB) and the Nederlandse Spoorwegen (NS) - the principal operators in Denmark and the Netherlands respectively.

All instances, except the DSB class, have been generated using a rolling stock optimizer. The events going in and out of the depots are extracted from the optimized schedule and define separate instances for each depot. Information about fleet size, train unit types, and depot track lengths are given by the railway operators.

The STOG class consists of twelve distinct rolling stock schedules obtained by optimizing the suburban railway network in the greater Copenhagen area (DSB S-tog). This gives up to twelve different event lists per station. Identical problem instances have been eliminated resulting in a total of 96 instances for the STOG class.

The DSB class consists of real-life data for a recurring weekly schedule at the busiest station in Denmark, which is located in the center of Copenhagen. Every day in the weekly schedule is unique thus resulting in seven instances for the DSB class.

The NS class consists of ten distinct rolling stock schedules for the whole country. This leads to ten different problem instances at eleven different stations. There are large differences between the event lists per station; some are large and some are small. Consequently, there are both difficult and relatively simple problem instances for the NS class.

The NS-HARD class is identical to the NS class, expect that fewer tracks are now available at busy stations. These artificial cases are therefore constrained in terms of capacity, in turn reducing the number of feasible parking plans. These have been included as an attempt to stress test the solution methods.

All computation experiments are performed on a dedicated machine equipped with two Intel(R) Xeon(R) CPU X5550 (2.67GHz) processors and 24 gigabytes of main memory. Version 12.6 of the commercial solver CPLEX is used to solve the MIP and CP based approaches. A time limit of 900 seconds is set for all experiments.

Class	Depot	Min	Max	Tracks	Length	Types
NS	AMR	157	159	9	2 267	4
NS	DDR	162	162	4	939	4
NS	EHV	153	179	20	7 061	4
NS	EKZ	97	97	5	1 590	4
NS	GVC	742	744	17	5 690	4
NS	HDR	82	82	3	1 143	4
NS	HFDO	561	561	8	3 020	4
NS	HN	75	75	12	2 023	4
NS	NM	268	268	25	6 495	4
NS	RTD	378	380	22	5 384	4
NS	ZP	87	87	9	4 127	4

Table 6.9: Continued summary of instances.

Class	Depot	Min	Max	Tracks	Length	Types
NS-HARD	GVC14	742	744	14	4 712	4
NS-HARD	HFDO5A	561	561	5	1 934	4
NS-HARD	HFDO5B	561	561	5	1 898	4
NS-HARD	HFDO6	561	561	6	2 197	4
NS-HARD	NM10	268	268	10	2 457	4
NS-HARD	NM11	268	268	11	2 657	4
NS-HARD	RTD11	378	380	11	3 085	4
NS-HARD	RTD12	378	380	12	3 410	4
NS-HARD	RTD13	378	380	13	3 669	4

Table 6.10: Continued summary of instances.

The following is a short summary of the solution methods proposed in Section 6.5 benchmarked in this section.

RMM A reference MIP approach solved using the CPLEX MIP solver.

DCMM A variant of the RMM where the pairwise order conflict constraints are generated *on-the-fly*.

CPM A constraint program formulation inspired by the composition model in [1]. The formulation is solved using the CPLEX constraint program solver.

CPMH A variant of the CPM where the number of different unit types assigned to the same track is limited.

RGCH A randomized greedy construction heuristic that is executed multiple times with different initial seeds.

CGM A column generation approach that assigns matching patterns to tracks.

TSM A two-stage decomposition method that solves the matching and parking problem in sequence using MIP approaches.

Before presenting the results in detail, we first note that the CGM, an extension of a method proposed in literature [2], is discarded from further analysis. The performance of this method on all instances was always inferior in comparison to the other methods. For small cases the time it took to produce an optimal solution to the linear programming relaxation was significantly greater than the time it took the MIP based approaches to produce a feasible solution. For the larger instances, CGM was unable to solve the root node relaxation (in a Branch-and-Price (B&P) framework) to LP optimality within the time limit in most cases. These results are consistent with

Class	No	RMM	DCMM	CPM	CPMH	RGCH	TSM
STOG	96	94	94	94	94	94	93
DSB	7	0	7	0	0	7	7
NS	110	0	93	84	101	110	110
NS-HARD	90	0	27	70	83	70	90

Table 6.11: Number of feasible instances found by the methods.

Class	RMM	DCMM	CPM	CPMH	RGCH	TSM
STOG	2	2	2	2	0	0
DSB	0	0	0	0	0	0
NS	0	0	0	0	0	0
NS-HARD	0	0	0	0	0	0

Table 6.12: Number of instances proved to be infeasible by the methods.

the study in [4], where a similar column generation approach was outperformed by a MIP approach for the parking problem only.

Table 6.11 and 6.13 show a comparison overview. Table 6.11 shows the number of instances for which a feasible solution is found per problem class per method within the time limit. Table 6.13 shows the average runtimes.

In the STOG class, which contains the smallest problem instances, 94 out of the 96 instances are feasible. All methods, except the TSM, were able to find the solutions. The TSM was unable to find a feasible solution for one of those instances. The average solution time is less than one second for all methods.

Solving the mathematical formulations of the other classes directly proves to be impractical. We observe that the RMM fails to solve all but the relatively small STOG problem instances. Significantly more cases can be solved by using the more efficient DCMM, CPM and CPMH variants. The DCMM can solve all DSB instances and a large portion of the NS instances, but only a few of the NS-HARD instances. The time limit becomes a prohibiting factor when using the DCMM. The CPMH is more successful at solving the NS and NS-HARD classes but unable to solve the DSB class due to the large number of unit types. The CPM performs relatively well compared to the RMM, but the performance is clearly dominated by the CPMH in terms of solutions found and average runtimes.

The CPMH, RGCH and TSM perform well on all the realistic instances as they are able to find the same number of feasible solutions. Further, RGCH and TSM are able to identify a feasible solution within a few seconds in average. However, for the artificial class of problem instances TSM proves to be the most efficient heuristic. The CPMH was unable to solve some of the larger instances (GVC), while the RGCH was unable to solve the more constrained instances (HFDO5A and HFDO5B).

Table 6.12 shows the number of instances for which infeasibility is proven per method within the time limit. First, we note that the heuristic methods RGCH is by definition not able to proof infeasibility. The two infeasible instances in the benchmark were detected by all exact approaches and the CPMH. The TSM was unable to prove infeasibility as at least one feasible matching exists.

Based on these results we can conclude that both the TSM and the RGCH method are very fast and efficient in finding feasible solutions. The CPMH is somewhat slower, but also successful in identifying feasible solutions in many cases. The RMM is clearly dominated by the DCMM,

Class	RMM	DCMM	CPM	CPMH	RGCH	TSM
STOG	0.5	0.4	1.3	0.6	0.0	0.0
DSB		255.8			0.1	1.3
NS		148.0	20.5	6.0	0.0	5.8
NS-HARD		267.5	78.2	33.3	0.3	1.1

Table 6.13: Average runtimes for finding solutions grouped by method

				DCMM			CPMH			RGCH	
Instance	#	F	Sub.	F	I	T	F	I	T	F	T
HFDO	10	10	20	20	0	0	20	0	0	20	0
GVC	10	10	20	4	0	16	20	0	0	20	0
RTD	10	10	30	30	0	0	30	0	0	30	0
DSB1	25	19	71	65	6	0	43	2	26	50	21
DSB2	25	18	71	64	6	1	40	4	27	50	21

Table 6.14: Summary of results achieved when running different methods on problem instances decomposed by splitting tracks and unit types. The columns respectively show the instance considered, number of decompositions, number of feasible decompositions, number of generated subproblems, and finally the number of **F**easible, **I**nfeasible and **T**imed-out instances for every method.

and the CPM by the CPMH. The DCMM solves fewer instances than the CPMH and requires more runtime, however it can, in contrast, solve some instances with a higher number of unit types.

Track Splitting

The problem can be decomposed into several independent problems by partitioning the full problem by unit types and tracks as described in Section 6.5.6. In this section we investigate whether this decomposition technique can improve the tractability of the exact methods.

We have randomly generated a number of partitions of a selected set of problem instances with the following procedure. First, a random number of groups is selected. Second, tracks and unit types are assigned randomly to the available groups. If any group has an empty set of unit types or tracks, then the whole generation is rejected. Further, it is ensured that the maximum depot capacity required by the unit types is less than the capacity of the tracks in the group. Finally, if units are positioned initially in the depot, then this naturally adds constraints to the generation of the groups.

The DCMM, CPMH and RGCH have been considered in this benchmark as they were unable to solve several instances in the previous section. The benchmark consists of large instances of the NS class that were unsolved by the RMM, DCMM and CPM. Decomposing this class of problems reduces the size of the underlying mathematical models significantly. Further, we consider two cases of the DSB class that were unsolved by the RMM, CPM and CPMH. A decomposition of these instances drastically reduces the number of variables and constraints needed by the CP model since the resulting number of different unit types is decreased.

An overview of the generated instances and results are shown in Table 6.14. The average runtimes are listed in Table 6.15. The HFDO, GVC and RTD instances were very successfully decomposed as all the resulting subproblems were feasible. The considered solution methods were able to solve these instances efficiently, except for the DCMM which was unable to produce a feasible solution for the subproblems of the largest instance. Nevertheless, DCMM is able to solve

Instance	No	DCMM	CPMH	RGCH
HFDO	10	83.4	13.3	0.1
GVC	10	3.2	9.9	0.0
RTD	10	18.0	1.7	0.0
DSB1	25	42.2	26.4	0.1
DSB2	25	32.6	30.8	0.1

Table 6.15: Summary of average runtimes achieved when running different methods on problem instances decomposed by splitting tracks and unit types. The columns respectively show the instance considered, number of decompositions and finally the average runtime for all found solutions.

Instance	Cases	DCMM	CPMH	RGCH	TSM
KH	3	2	3	3	1
FM	3	3	3	3	0
EHV	3	3	3	3	3
HDR	3	3	3	3	0

Table 6.16: Summary of results achieved when running different methods on problem instances with overnight parking. The columns respectively show the instance considered, number of problem instances, and the number of solved instances for every method.

more instances using this decomposition technique. The DSB1 and DSB2 instances were on the contrary decomposed into both feasible and infeasible subproblems. The DCMM is able to solve all subproblems, except one, efficiently. The CPMH is now able to solve more than half of the subproblems but several remained unresolved. This is an improvement compared to the non-decomposed results. Interestingly, decomposing the problem proved unhelpful for the RGCH as many feasible subproblems were left unsolved. The RGCH was able to solve the original instances of the problems. A reduction in computational time is observed for both the DCMM and the RGCH in Table 6.15.

Overnight Parking

In the common case all units leave the depots during the early hours and enter the depot at the end of the day. Some units enter and leave the depots during the day, e.g. before and after rush hour periods. The capacity at depots is therefore not very limiting during the day, which makes it easy to plan this intermediate period.

The considered problem instances do not stipulate any particular parking order at the end of the day. Consequently, no ordering conflicts arise regardless of the final track assignment, when the depots are close to being at capacity. Realistically, a smooth transition from one day to another is desirable, making sure that units can leave the depot in a conflict free manner the following day. In our final benchmark, we combine the planning instances to include the events for two days of operation, thus forcing the solution methods to consider the overnight parking.

Tables 6.16 and 6.17 show the results of the overnight parking instances. The instances are naturally larger than the original ones, and require more time to solve as two days of events have been combined. The results show that the considered methods, except the TSM and DCMM, can efficiently solve all instances. All methods except the TSM integrate matching and parking, where the RGCH does this by trying many different matchings in the solution construction. The TSM first finds one feasible matching and tries to park the matched events. Fixing the matching in a early stage does, however, restrict the flexibility when resolving the ordering conflicts. In contrast to the other benchmarks, these problem instances contain at least one very busy period,

Instance	Cases	DCMM	CPMH	RGCH	TSM
KH	3	449.5	223.0	0.1	12.3
FM	3	8.2	1.1	0.0	0.0
EHV	3	244.4	11.9	0.1	0.4
HDR	3	17.7	1.0	0.0	0.1

Table 6.17: Summary of average runtimes achieved when running different methods on problem instances with overnight parking. The columns respectively show the instance considered, number of problem instances, and average runtimes.

where the depots are close to being at capacity. Evidently, an approach that fixes, i.e., only considers one matching, may very well fail to find a feasible solution. The average runtimes shown in Table 6.17 reveal that the CPMH is faster than the DCMM in general when considering these extended instances. We note, however that these instances only contain 2-4 different unit types. The performance of the CPMH is expected to decrease with higher numbers of different unit types.

6.7 Conclusion

In this paper we have developed and benchmarked different models and solution approaches to solve the Train Unit Shunting Problem. Given a feasible rolling stock circulation, the objective of the solution approaches is to find a valid matching and shunting plan. A number of computational experiments have been performed on multiple problem instances from three different railway operators. The benchmark highlights strengths and weaknesses of the considered approaches. A platform parking extension is described, as platform tracks are currently used for overnight parking in some railway operations.

The main benchmark, consisting of multiple daily problem instances, revealed the main weakness of the exact models based on mathematical models, i.e. RMM and CPM. These approaches were outperformed by the other approaches. The resulting mathematical models quickly consumed more than 24 gigabytes of memory due to the large number of constraints and/or variables required. Using delayed constraint generation (for the RMM) the DCMM is able to solve significantly more instances. The heuristic extension CPMH method (of CPM) further outperforms the DCMM when considering the instances with a small number of different unit types. In turn, the DCMM can solve the instances with a high number of unit types. On average, the solved instances were solved in a few minutes by these methods.

A randomized construction heuristic, the RGCH, was able to solve almost all instances within one second. However, some harder and artificially generated instances were left unsolved by the RGCH. In the main benchmark the non-integrated method TSM proved to be most successful, solving all but one of the feasible instances within a few seconds. Ironically, the unsolved instance was a relatively small problem instance. Finally, a column generation approach, the CGM, was also considered, however, it was not benchmarked as it was always inferior to the MIP/CP based approaches on small instances and required too much computation time to solve larger instances.

The TUSP can be decomposed by splitting the available tracks and unit types into several independent and smaller subproblems. A set of the larger instances were decomposed in a second benchmark. In general, most of the the resulting partitions were feasible. Using this decomposition the DCMM and CPMH were able to solve more problem instances than before, but not all. In fact, the RGCH performed worse than before as it was unable to find solutions for some of the problems that were solved originally.

In a final benchmark a number of instances were combined in order to solve two days of operation. Interestingly, the results show that no added difficulty is introduced when using the integrated methods. However, the TSM is now unable to solve most of the problem instances.

The considered solution approaches have both strengths and weaknesses. The results show that no method is superior. Solving the full mathematical formulations, i.e. RMM, CPM, directly proves to be ineffective. The DCMM is able to prove infeasibility in most cases. In addition, note that the proposed feasibility checks in Section 6.4 are very efficient - few instances were infeasible in general. Very fast solutions can be found using the RGCH but it proves to be inefficient for the more constrained instances. Finally, the TSM solves more instances using a few seconds, but has the drawback of using a fixed matching, which can lead to premature infeasibility. In conclusion, given the low runtime requirement of the RGCH and the TSM, these approaches form a reasonable choice as a first step in any solution framework. If no solution is found the DCMM and the CPMH can be adopted in a subsequent phase.

This paper focusses only on finding a feasible solution. There is no distinction made between any feasible solution. In future research it might be interesting to extend the models by considering an objective in order to find a solution with, for instance, homogeneous tracks. Furthermore, several important restrictions have not been considered. If, for instance, the rolling stock circulation passes our feasibility check, it might still be infeasible with respect to the available crew members present for shunting operations at a station. Consequently, the crew has to be taken into account in the TUSP in future research. Other practical aspects needs to be taken into account as well, e.g. parking whole compositions instead of single units, units might require maintenance at the station, and cyclic rolling stock circulations. In future research it would be interesting to include some or all of these aspects in the TUSP.

Bibliography

[1] Pieter-Jan Fioole, Leo Kroon, Gabor Maroti, and Alexander Schrijver. A rolling stock circulation model for combining and splitting of passenger trains. *European Journal of Operational Research*, 174(2):1281–1297, 2006.

[2] Richard Freling, Ramon M Lentink, Leo G Kroon, and Dennis Huisman. Shunting of passenger train units in a railway station. *Transportation Science*, 39(2):261–272, 2005.

[3] Michael R Garey, David S Johnson, and Larry Stockmeyer. Some simplified np-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63. ACM, 1974.

[4] Jørgen Thorlund Haahr, Richard Martin Lusby, Jesper Larsen, and David Pisinger. *Simultaneously Recovering Rolling Stock Schedules and Depot Plans Under Disruption.* Proceedings of the 13th Conference on Advanced Systems in Public Transport (CASPT), 2015.

[5] R Haijema, CW Duin, and NM Van Dijk. Train shunting: A practical heuristic inspired by dynamic programming. *Planning in Intelligent Systems: Aspects, Motivations, and Methods*, pages 437–475, 2006.

[6] Per Munk Jacobsen and David Pisinger. Train shunting at a workshop area. *Flexible services and manufacturing journal*, 23(2):156–180, 2011.

[7] Richard M Karp. *Reducibility among combinatorial problems.* Springer, 1972.

[8] Leo G Kroon, Ramon M Lentink, and Alexander Schrijver. Shunting of passenger train units: an integrated approach. *Transportation Science*, 42(4):436–449, 2008.

[9] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[10] Ramon M Lentink, Pieter-Jan Fioole, Leo G Kroon, and Cor van't Woudt. Applying operations research techniques to planning of train shunting. *Planning in Intelligent Systems: Aspects, Motivations, and Methods*, pages 415–436, 2006.

[11] Marco Lubbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, 53:1007–1023, 2004.

[12] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[13] D. Pisinger. *Algorithms for Knapsack Problems.* PhD thesis, DIKU, University of Copenhagen, Denmark, 1995. Technical Report 95-1.

[14] Thomas Winter. *Online and Real-Time Dispatching Problems.* PhD thesis, Technical University, Braunschweig, Germany, 1999.

Chapter 7

Integrated Planning of Rolling Stock Schedules and Shunting Yard Plans

Jørgen T. Haahr* Richard M. Lusby*

*Department of Management Engineering, Technical University of Denmark,
Produktionstorvet, Building 426, DK-2800 Kgs. Lyngby, Denmark
jhaa@dtu.dk, rml@dtu.dk

[1] **Abstract** In this paper we consider a framework to integrate two important railway optimization problems: the Rolling Stock Scheduling problem and the Shunting Yard problem. We present two similar solution approaches that solve this integrated problem, both based on a branch-and-price methodology. Furthermore, we provide a comparison of several solution strategies to the Track Assignment Problem, a subproblem of one of the solution approaches, and show the heuristic nature of a previously proposed *optimal* approach by presenting a counter-proof. We analyze the performance of the proposed solution approaches on a number of artificial data sets as well as several real-life case studies provided by DSB S-tog, a suburban train operator in the greater Copenhagen area. The computational results show that the solution approaches are eligible for short-term planning. High quality solutions for the integrated problem instances are typically found within a few minutes.

7.1 Introduction

A passenger railway operator must exercise careful planning in order to make detailed schedules for the day-to-day operations. A railway line plan stipulates the underlying structure of the timetable, which in turn needs to be serviced by the fleet of rolling stock and personnel of train drivers. Identifying a high quality solution for the railway operation has proven to be a difficult and complex optimization problem. The current practice is to decompose the problem into smaller subproblems and solve them sequentially. Two apparent drawbacks to this approach are the degradation of overall solution quality and the infeasibility occurring due to poor decisions made in previous stages; e.g. a timetable with a valid rolling stock schedule may be infeasible with respects to the shunting yard problem. The latter drawback is magnified in a short-term or real-time setting, where the time to react is limited. Integrating the planning problems remedies the

[1]Submitted to *Transportation Research Part B*, October 2015

need for iterating through infeasibility and the generation of sub-optimal solutions. Due to the size and complexity of each of these problems, not to mention the short reaction time available to find a solution, it is not surprising that there has recently been an increased interest in computer aided decision support, or disruption management tools that assist planners in recovery operations within the railway industry, see e.g. [19]. A study in the airline industry demonstrates that integrating aircraft routing and crew pairing results in cost reductions (in addition to obtaining more robust solutions) when compared to the solution used in practice [29].

Planning the rolling stock and depot movements sequentially in separate phases can lead to infeasible or sub-optimal solutions. A depot infeasibility would most likely be due to an ordering violation on one of the available tracks, i.e. two units cannot enter and exit the depot in the order prescribed by the rolling stock plan. Often such infeasibilities manifest themselves in railway networks with scarce depot capacity. The Copenhagen Suburban Railway Operator (DSB S-tog) reports that an optimized rolling stock usually is infeasible with respect to the shunting yard movements, thus requiring manual repairs in order to obtain an overall feasible solution.

In this work we consider two important scheduling problems: the Rolling Stock Scheduling Problem (RSP) and the Shunting Yard Problem (SYP), respectively. The main focus entails integrating these problems, which we term the Integrated Rolling Stock and Shunting Yard Problem (IRSP). To the best of our knowledge, an approach integrating these two problems has not been previously proposed. The strategic and tactical level planning variants of each problem have been studied independently in the literature [6, 20, 8], and models and methods are also available for the rolling stock rescheduling problem [12, 22]. However, research that explicitly addresses shunting yard planning, not to mention its integration with rolling stock scheduling, seems to be noticeably absent. We present an optimization based framework, which extends previous work [6, 12], for simultaneously solving both problems in an integrated framework. Rolling stock scheduling necessitates allocating train units to the timetabled trips in such a way that the overall cost of executing the schedule is minimized, while at the same time ensuring a sufficient number of seats are provided for the passengers. Additional constraints such as balancing the available units must also be respected, e.g target inventory levels for each of the unit types at each of the shunting yards. Finding optimal or high-quality rolling stock solution is the focus of the RSP. Shunting yard planning, such as the Train Unit Shunting Problem (TUSP) and Track Assignment Problem (TAP) [8] or the SYP on the other hand, primarily focuses on the train units not in service. In order to cover the expected passenger demand as closely as possible, the composition of train services can be changed over the course of the planning horizon by coupling or uncoupling rolling stock units. Furthermore, units must routinely be taken out of service for maintenance and cleaning purposes. A shunting yard can be considered as a depot, usually adjacent to a railway station, for storing rolling stock that are not in service. It usually consists of a number of parallel storage tracks on which units can be parked. Which track and in which order to park such units is the focus of the SYP. Each depot track has a certain capacity, limiting the number of units that can be assigned the track at any one time. Furthermore, most depot tracks can only be accessed from one end, implying that they function as last-in first-out (LIFO) stacks. In a few cases, tracks may be accessible from both ends.

Two variants of a framework that solves the integrated problem are proposed. The first variant extends the Branch-and-Price (B&P) framework of [12], which schedules rolling stock units, to also include the identification of parking plans for each of the depots in the network. In [12] depot capacity is only implicitly considered in the form of an aggregated constraint on the total length of track available. It does *not* consider any ordering conflicts that could arise, nor if it is possible to park the units movements when individual track capacities are considered. The model is a unit based decomposition which generates trajectories for each of the units in the fleet and in doing so identifies compositions for each of the trains that will be run. The underlying model ensures when coupling or uncoupling units at a given depot, the resulting composition change is possible. From the specific individual unit trajectories it is also possible to determine at which times a given unit enters, or exits, a given depot. Utilizing this information, we extend the B&P method of [12] to a

Branch-and-Price-and-Cut (BAPC) approach, where on finding a feasible solution to the routing problem, we test the feasibility of all shunting yard movements in the network. If feasible, the solution is accepted, otherwise it is cut away and a new routing solution is generated. We refer to this variant as the Route First Method (RFM), as units are routed before checking shunting yard feasibility. The second proposed variant is similar. A flow-based Mixed Integer Program (MIP) inspired by [6] is adopted and extended to a Branch-and-Cut (BAC) approach, where encountered rolling stock solutions are tested. If the resulting depot movements have a feasible matching and parking plan at all individual depots, then the solution is accepted. Otherwise, if any one depot is infeasible then the solution is cut away. This variant is referred to as the Route Last Method (RLM) since unit routes can be determined after finding a feasible rolling stock and shunting yard solution. We note that in contrast to the RFM, a generalized version of the TAP constitutes the separation routine in the RLM.

This paper makes the following important contributions to literature on railway rolling stock scheduling and shunting yard planning:

- Two variants of an integrated BAC frameworks for scheduling rolling stock and depot plans are proposed and benchmarked.
- A comparison of methods for solving the TAP. We propose a BAC approach for a MIP, extend an existing [8] heuristic column generation approach to full B&P framework, and present an optimal B&P framework.
- A counter example is given that highlights the heuristic nature of a previously proposed optimal method in [8] for generating the optimal Linear Program (LP) solution to the TAP.
- A heuristic swapping routine that can be used to repair a depot plan as an alternative to cutting the infeasible solution away.

This research is carried out in collaboration with DSB S-tog. All depots on this network consist of LIFO tracks and depot space is at a premium. Finding a feasible rolling stock schedule is only part of the problem since a feasible parking plan to facilitate the required shunting movements may not exist. We test the performance of the algorithms on several real-life case studies provided by DSB S-tog.

This paper is structured as follows. In Section 7.2 we provide a brief summary of the relevant literature, where any differences to existing approaches are described. Section 7.3 then introduces a more formal description of the RSP and the SYP. We introduce mathematical models for the respective problems in Section 7.4. Section 7.5 discusses solution methods for the presented models and describes in detail the complete, integrated framework. It is in this section that we provide a counter example to the proposed optimal approach of Freling et al. [8] for the TAP. In Section 7.6 the performance of the proposed methodology is analyzed on both an artificial and a realistic set of problem instances. Finally, conclusions and future research directions are outlined in Section 7.7.

7.2 Literature

In this section we review relevant literature in the field of RSP and the SYP. To the best of our knowledge, no research has been published that integrates both problems. For the RSP we include models and methods used for solving the tactical and short-term variants. For the latter, however, to our knowledge no studies dedicated to short-term planning of shunting yards exist. The TAP is considered a specialization of the SYP, where matchings are assumed to be given. The ordering restrictions inherent in the TAP or the SYP are not unique to passenger rail, nor trains for that matter. We draw analogies with bus depot planning and freight rail car classification.

Over the past few years several different models have been put forward for solving rolling stock problems, both from the planning and the rescheduling perspective. An approach for determining the minimum circulation of train units required to operate a timetable is described by Schrijver [26]. A more elaborate MIP model that can handle the combining and splitting of trains is proposed by Fioole et al. [6]. This model forms the basis of [23], where a rolling horizon framework is developed for solving a rolling stock rescheduling problem. These three approaches can all be classified as anonymous unit flow models. As such, one is unable to track individual unit movements. As an alternative, Haahr et al. [11] present a path based formulation for the same problem and solve this using a B&P framework. Each path specifies an itinerary (or route) for a particular unit type through the network. Train compositions are handled by the master problem, making sure that any composition changes are legal. Note that the model does not assign physical unit routes but sets of routes that are consistent with the unit types available. This methodology does therefore not deal with anonymous units and is equally applicable at the tactical planning level.

Not surprisingly, a number of studies concerning variants of the SYP have also been conducted. The problem of dispatching trams from a storage yard is addressed by Winter and Zimmerman [30]. In order to achieve a departure order satisfying the scheduled demand, several shunting operations may be necessary. The authors describe combinatorial optimization models as well as both exact and heuristic approaches for solving the real time dispatch problem. Scheduling trams in the morning is also the topic of Blasum et al. [2]. The authors prove the NP completeness for the problem of finding an assignment of parked trams (of different types in stacks) to departure times without any additional shunting movements. A graph theoretical approach to shunting train units in a railway depot is adopted by Di Stefano and Koči [27]. The complexity of several subproblems is studied and the objective of all problems is to minimize the number of tracks needed to park the units. Three heuristic approaches to train shunting in a workshop area are described by Munk and Pisinger [18], while Føns [7] considers different mathematical models and approaches for the SYP. Both latter studies consider cases arising at the Copenhagen suburban railway operator, DSB S-tog.

The problem of dispatching buses from a bus terminal is the focus of Gallo et al. [9]. Initially the problem is formulated using the Quadratic Assignment model approach of [30]. In addition, the authors also describe a new model that takes into account the fact that the buses can have different lengths. The resulting model is shown to be well suited to decomposition, and hence the authors present a Lagrangian decomposition based approach. Real-life instances from the Florence Public Transportation Company are studied. It is concluded that the algorithm can find good quality solutions at low computational cost. Other bus parking related research includes Hamdouni et al. [14, 15]. The former study considers identifying robust parking solutions and argues that one should generate solutions in which the bus lanes have at most two different types. Having fewer types on a lane reduces the potential for ordering (or crossing) conflicts, possibly at the expense of wasting parking capacity. The latter study introduces a Benders decomposition approach for minimizing bus type mismatches between arrival and departure pairs. That is, in the problem considered it is possible to supply a bus of a different type to that which is demanded, but at a cost. Having as few bus types on a given parking lane is also a priority.

In [8], a model for a variants of the integrated problem is proposed, termed the TUSP, however, the solution method decomposes the problem into a matching problem and a TAP. The matching is solved first and then used as input to the TAP. The authors propose a MIP model for the matching problem, while the parking problem is solved using a column generation approach. Computational experiments focus on two case studies at station Zwolle in the Netherlands. It is not completely clear if a column generation approach actually outperforms a MIP approach for the instances considered. Furthermore, we argue, by presenting a counter example that the dominance criteria stated is indeed not exact, i.e. it can potentially remove the optimal solution.

A similar decomposition is also suggested by Lentink et al. [21]. However, two additional steps focusing on the routing of the train units from their arrival platforms to their designated depot tracks are also included. Between the matching of arrivals with departures and the parking of

the units, an estimate on the cost of routing the trains is calculated. These costs are then used when solving the parking problem. Upon parking the units, actual routes from the platform to the depot tracks are obtained. A heuristic that sequentially routes the units is devised, and the complete routing solution is improved using a 2-opt heuristic.

In [13] the authors propose a dynamic programming based heuristic for solving a variant of the SYP. A realistic test case from the railway station Zwolle in the Netherlands is used to test the developed methodology. The test case considers a 24 hour period during which 45 units arrive and 55 units depart. The depot has 19 tracks with a total length of 4000m. The simple heuristic is fast and flexible and produces promising results; however, only a single instance from real-life is considered.

The work of Kroon et al. [20] extends the work in [8] and presents a fully integrated model for solving both the matching problem as well as the parking problem. A large MIP model is proposed that attempts to minimize the number of split compositions and the number of different unit types simultaneously parked on the same depot track. The authors indicate how the model can be strengthened through the addition of clique inequalities and also how to model more practical restrictions. For example, a discussion on how to deal with trains composed of several train units as well as how to model depot tracks that can be approached from both sides is included. The concept of a virtual shunting track is introduced in order to help identify which tracks should be heterogeneous and which tracks should be homogeneous from a unit type perspective. Computational experiments focus on two Dutch stations and consider up to 125 train units of 12 different types that need to be parked.

An advanced planning tool for shunting operations is described by Fransoo et al. [28]. Like almost all previous work on the SYP, this approach first solves a matching problem that determines how arriving units are matched to departing units. This is done via a network flow algorithm. A k-shortest path algorithm is used to assign a track to each unit, while a modified version of an undirected k-shortest path is used to route the units. Finally the approach also assigns drivers to shunt operations.

Aside from passenger railway operators, ordering problems on storage tracks are highly prevalent in the freight rail industry. In order to reach their final destination freight rail cars are sorted at so-called *classification* yards where incoming trains are sorted into blocks of rail cars that share the same destination. The blocks are subsequently combined to form outbound trains. The order in which to process inbound trains, which blocks to build, which track to assign a block, and how to build outbound trains, are all decisions that need to be made. For a survey of shunting in the freight rail industry, the reader is referred to e.g. [10] and [3].

To summarize, when solving the SYP one first typically solves a matching problem before solving a parking (and possibly a routing) problem. This stems from the fact that rolling stock allocation is typically done using MIP models that generate anonymous unit routes. In what follows, we present two variants of approaches that integrate the rolling stock scheduling and depot planning. In the RFM rolling stock itineraries are generated as part of the rolling stock scheduling before testing depot feasibility, while the RLM does the same except it leaves the rolling stock routing part to last.

7.3 Problem Description

The main problem deals with the integration of two scheduling problems. This section defines the two underlying subsections and provides a detailed description of each in turn. Section 7.3.1 introduces the RSP, while Section 7.3.2 is devoted to a discussion on the SYP.

7.3.1 Rolling Stock Scheduling

Given a timetable the rolling stock scheduling problem entails allocating a fleet of units, possibly of different types, to the timetabled trips. The trips each unit can be assigned to, depend on the unit's current location and potentially also its type. A unit's type specifies the characteristics of the unit, i.e. its length, operating cost, and capacity. Typically, one tries to allocate the fleet in such a way that sufficient seat capacity is provided for the passengers; however, minimizing the number of additional cancellations and/or operating costs might also be a preferred objective. Robustness aspects also affect a rolling stock plan, e.g., by minimizing the number of coupling and uncoupling operations and by requiring a minimal amount of time to perform these operations. The end-of-day balance guarantees that a certain number of units of each type are available at each of the depots at the end of the day. This ensures that the rolling stock operations the following day can start as planned. Other constraints include the current location of the unit types, the maximum length of train compositions on trips, and aggregated depot capacities. In reality shunting yards consist of multiple parallel tracks of different lengths. Solely respecting the aggregated length does not imply that units under consideration can in fact be parked on the discrete set of tracks. Furthermore, this does not rule out the possibility for ordering violations on the depot tracks when considering all tracks as well as the exact timings of unit movements into and out of the depot.

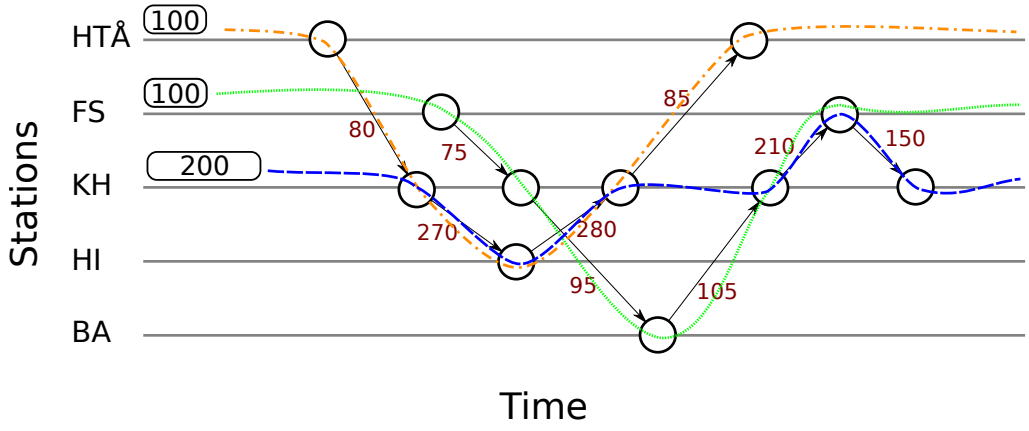


Figure 7.1: An example of a rolling stock problem instance using a time and space diagram. Stations with shunting yards are depicted vertically, some of which contain initial rolling stock units of certain capacities. Arcs represent train trips that run from one station to the next. The number associated with each arc specifies the passenger demand. The colored paths illustrate the itineraries taken by the available rolling stock units.

A simplified example of a rolling stock problem instance and solution is shown in Figure 7.1. The solution is cyclic, as units end at the station where they were originally positioned. All passenger demand is covered by coupling and uncoupling the rolling stock units at key positions. For example, a small and large unit are coupled at `KH` in order to satisfy the demand of 270. It is shortly after uncoupled and coupled again to a different trip. Note, that whenever a unit is not following some train trip, it is out of service, and has to be parked in a depot.

7.3.2 Shunting Yard Planning

The SYP can be stated as follows. Can the set of scheduled activities, induced by a rolling stock schedule, be feasibly parked in all shunting yards? In the general version of the problem, the activities specify entering and leaving unit types, not physical units. A matching between physical units, arrivals and departures is therefore also included in the problem. A problem instance is

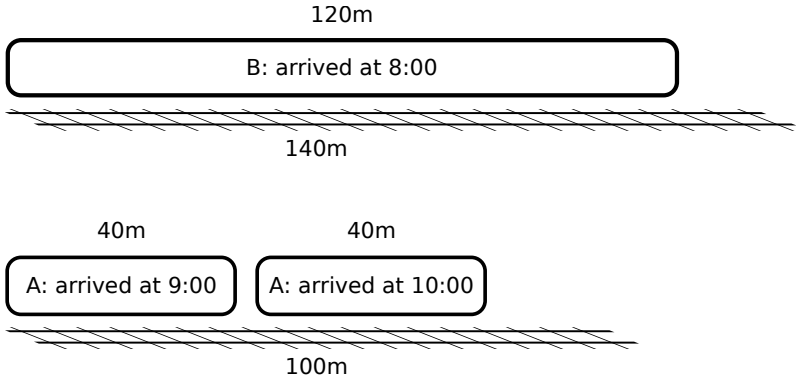


Figure 7.2: Shunting yard problem example. Three units of two distinct types are parked on two tracks.

feasible if both a feasible matching and corresponding parking plan exist. We argue that in a short term planning, it is sufficient to detect feasibility of the depots as the majority of the cost is incurred by the rolling stock schedule. Note, if feasible, depots can always be resolved independently to achieve a better parking. One key difference between different planning phases is whether an initial fleet position (on the depot tracks) is given, or whether it must be found.

As mentioned earlier, the SYP focuses on a specific depot and tries to assign unit movements in such a way that each unit movement into and out of the depot can be performed in a conflict free manner and such that no depot track capacity is violated. This requires solving two smaller sub-problems known as the *Unit Matching Problem* and the TAP. The first is an optimization problem which matches arrivals with departures and assigns each a physical unit. This is required as the rolling stock allocation is typically anonymous in a rolling stock schedule. In other words, only the type of unit required on a certain trip is known, not which actual unit which will perform the trip. Rolling stock units are typically considered interchangeable if the types are identical.

For example, consider a shunting yard with two tracks of lengths 100m and 140m. Three arrivals are scheduled, two units of type A (of length 40m) arrive at 9:00 and 10:00, and one unit of type B (of length 120m) arrives at 8:00. Three departures are scheduled, a unit of type A must leave at times 12:00 and 16:00, and a unit of type B must leave at 20:00. Figure 7.2 depicts the shunting yard at time $11:00$. Up to 11:00, the only feasible parking plan is shown in the figure, any other assignment is infeasible due to the length of the tracks. Only two feasible matching exists as the unit types must match, but both matching assignments are not feasible. The unit arriving at 10:00 must be assigned to the 12:00 departures, otherwise the movements of the A units would be conflicting.

7.4 Models

In this section we describe models used for solving the SYP and RSP, which serves as a foundation for a separation routing in the integrated frameworks for solving the IRSP. First we present a compact MIP formulation of the SYP in Section 7.4.1, which can be solved using a commercial solver. A reduction of this formulation can be used for the TAP as it forms a special case of the SYP. Second, we present and discuss the column generation approach presented by Freling et al. [8]. We extend this work by developing a full B&P framework. In Section 7.4.3, we present the MIP model that is used for solving the RSP.

7.4.1 A Compact Matching and Parking Formulation

When solving the general SYP, we are given a set of tracks, $\mathcal{T}$, of different lengths in addition to a set of arrival and departure events, $\mathcal{E}_{arr}$ and $\mathcal{E}_{dep}$ respectively. The set is ideally deduced from an optimal solution to the RSP by observing when unit types are coupled to and uncoupled from train services. Associated with each event $e \in \mathcal{E}_{arr} \cup \mathcal{E}_{dep}$ is information concerning a time $time(e)$, length $len(e)$ and type $type(e)$. We note that any initial parking plan can be modeled by adding artificial events before the planning horizon and fixing the track assignment for these. The position of a unit declares not only the track, but also the index of the unit in the stack of units initially parked on the track. Without loss of generality we can assume that the number of arrivals and departures are equal in size, by generation of artificial events.

Every arrival can be matched to one of possibly many departures, given that the unit type of the events is the same. We define a set of all possible matchings:

$$\begin{aligned}\mathcal{M} = \{(e_a, e_d) | e_a \in \mathcal{E}_{arr} \wedge e_d \in \mathcal{E}_{dep} \\ \wedge type(e_a) = type(e_d) \\ \wedge time(e_a) \leq time(e_d)\}\end{aligned}$$

An arrival and departure event present a valid matching if and only if the unit type is identical and the departure occurs after the arrival in time. Note, that the set of matchings should in practice be restricted further, e.g. by requiring a minimum separation time between the arrival and departure. We define $\mathcal{M}_e \subset \mathcal{M}$ to contain all matchings that include event e, and c_{mt} to define the cost or unattractiveness of parking matching $m \in \mathcal{M}$ on track $t \in \mathcal{T}$.

In the formulation for solving the SYP variables x_{mt} are introduced and indicate whether or not matching $m \in \mathcal{M}$ is assigned track $t \in \mathcal{T}$. A value of 1 indicates that the corresponding arrival and departure are matched, and in addition the arriving unit is parked on track t in the shunting yard. Any pair of matchings that cannot be parked on the same track due to LIFO violations are said to be in conflict, and all such pairs are contained in the set $\mathcal{C}$. The length of track $t \in \mathcal{T}$ is denoted by $L_t \in \mathbb{R}$. Further, let $\mathcal{N}_e$ define the set of matchings, that would be present in the depot upon arrival of event $e \in \mathcal{E}_{arr}$. The full formulation can now be stated as follows.

$$\text{Minimize:} \sum_{m \in \mathcal{M}} \sum_{t \in \mathcal{T}} c_{mt} \cdot x_{mt} \tag{7.1}$$

$$\sum_{t \in \mathcal{T}} \sum_{m \in \mathcal{M}_e} x_{mt} = 1 \qquad \forall e \in \mathcal{E}_{arr}, \tag{7.2}$$

$$\sum_{t \in \mathcal{T}} \sum_{m \in \mathcal{M}_e} x_{mt} = 1 \qquad \forall e \in \mathcal{E}_{dep}, \tag{7.3}$$

$$\sum_{m \in \mathcal{N}_e} len(m) \cdot x_{mt} \leq L_t \qquad e \in \mathcal{E}_{arr}, t \in \mathcal{T}, \tag{7.4}$$

$$x_{m_1 t} + x_{m_2 t} \leq 1 \qquad \forall (m_1, m_2) \in \mathcal{C}, t \in \mathcal{T}, \tag{7.5}$$

$$x_{mt} \in \{0, 1\} \qquad \forall m \in \mathcal{M}, t \in \mathcal{T}, \tag{7.6}$$

The objective (7.1) is to minimize the overall cost of the found solution. Constraints (7.2) and (7.3) require that every arrival and departure event are selected in exactly one matching. Every event must be covered exactly once. Constraint set (7.4) ensures that the track capacity is never violated by ensuring that the capacity is respected on each track at the arrival of a new unit. Constraints (7.5) stipulate the LIFO restrictions; there is a pairwise packing constraint included for any two events in conflict. Finally, variable domains are given by (7.6). Note that constraints (7.5) can be strengthened by identifying stronger clique inequalities. Typically this set

of constraints can be problematic as the formulation could become prohibitively large if there are many conflicts.

The TAP is a special case of the presented problem, where the matching is predetermined, i.e. exactly one matching exists for every arrival and departure. A similar formulation defines this problem where either Constraints (7.2) or (7.3) can be omitted, and replaced by a cover of all pre-matched events instead. This formulation is similar in structure to the MIP formulations given in [21], and [20]. This approach focuses on the events and attempts to determine the best assignment of events to tracks. Note that we define the problem to have an objective function here, despite the fact we solve it as a feasibility problem.

7.4.2 A Column Generation Model

We introduce a second formulation for the TAP similar to that of the compact formulation; however, the cover and track capacity constraints are enforced in the construction of the variables and are thus not explicitly needed in the formulation. This formulation has been proposed by, among others, Freling et al. [8] and Føens et al. [7].

In this approach one looks at identifying *parking plans* for each of the tracks, and is the formulation obtained by applying Dantzig-Wolfe Decomposition to the TAP formulation variant above, where the track capacity constraints, and match-cover and LIFO constraints are placed in the subproblem and thereby implicitly satisfied in the construction of the variables. A parking plan for a given track $t \in \mathcal{T}$ simply refers to a set of events that can all use track t in a conflict-free manner without violating the track capacity. Note, that with the absence of a matching problem, an event now refers to the arrival and departure of a rolling stock unit. The set of events, $\mathcal{E}$, refers to all unit activities that occur in the depot.

We denote the set of all parking plans for $t \in \mathcal{T}$ as $\mathcal{P}_t$ and introduce a new set of binary variables x_{tp} that govern the selection of a particular track plan $p \in \mathcal{P}_t$ for depot track $t \in \mathcal{T}$. A cost c_{tp} is associated with each such variable and indicates the unattractiveness of the assignment; this cost is merely the sum of the costs of each of the events contained in the parking plan. We define y_e variables, one for each event, to include the possibility of not covering an event. Like, the compact formulation, each is assigned a large penalty cost M. Finally, the binary parameter a_{ep} is used to indicate whether event $e \in \mathcal{E}$ is included in track plan $p \in \mathcal{P}_t$ or not. The full formulation is given below.

$$\text{Minimize: } \sum_{t \in \mathcal{T}} \sum_{p \in \mathcal{P}_t} c_{tp} x_{tp} + M \sum_{e \in \mathcal{E}} y_e \tag{7.7}$$

$$\sum_{p \in \mathcal{P}_t} x_{tp} = 1 \qquad \forall t \in \mathcal{T}, \tag{7.8}$$

$$\sum_{t \in \mathcal{T}} \sum_{p \in \mathcal{P}_t} a_{ep} x_{tp} + y_e = 1 \qquad \forall e \in \mathcal{E}, \tag{7.9}$$

$$x_{tp} \in \{0, 1\} \qquad \forall t \in \mathcal{T}, p \in \mathcal{P}_t, \tag{7.10}$$

$$y_e \in \{0, 1\} \qquad \forall e \in \mathcal{E}. \tag{7.11}$$

An interpretation of this model is as follows. The objective function, given by (7.7), is identical in structure to that of Objective (7.1); however, the aim here is to find a minimum cost selection of track plans. Constraints (7.8) ensure that each depot track is assigned exactly one of its possible track plans (including the null assignment, i.e. a parking plan containing no events), while Constraints (7.9) enforce the restriction that each event must be present in exactly one of the track plans, or left uncovered. Variable domains are stated by (7.10) and (7.11). While this

formulation has a constant row dimension (independent of the number of conflicting events), it can have an exponential number of variables; for large problems there could be many possible parking plans. Hence, a column generation approach is typically the preferred solution method, generating only variables that have the potential to improve the objective function. This approach was first described in the proposed method of [8], where it is coupled with a heuristic Branch-and-Bound (B&B) strategy. In Section 7.5 we outline a full B&P procedure, and correct an error in the proposed column generation procedure.

Unit Route Swapping

Feasibility is unlikely to be a problem if the number of depot tracks is sufficiently large; however, for DSB S-tog, depot capacity is a scarce commodity. If a TAP problem instance is infeasible, feasibility can be restored if it is possible to swap the routes of two events in conflict. The given fixed matching in the TAP is to some degree now relaxed as the departure of two units can be swapped. Swapping can only be done if the units are considered interchangeable, i.e. are of the same type and are compatible with respect to other unit constraints. With the absence of other constraints in this paper, such as maintenance, all units of same type are interchangeable. At railway operators, such as DSB S-tog, maintenance checks on units are issued at certain times and routes are allocated to units whilst adhering to this. Thus, swapping units is not considered a general remedy to correct for feasibility. Note that in this work, we do not take route nor maintenance into account, but we assume this is done in a post-processing phase.

7.4.3 Rolling Stock Model

A MIP formulation is presented in this section that defines a valid rolling stock schedule. We adopt the formulation presented by Fioole et al. [6]. The model forms the basis of the integrated framework described in Section 7.5.4. The rolling stock formulation specifies which train composition is assigned to every trip stipulated by the timetable. A composition defines a number of coupled units, in a specific order. The assigned compositions are interdependent as the compositions of two consecutive trips must adhere to both physical and operational rules, e.g. a station may only allow one unit to be coupled to or uncoupled from the back of the train, and turning at some station forcibly reverses the composition for the next trip. The solution of this model ultimately specifies the flow of unit types over all trips. We refer to Fioole et al. [6] for a complete description of the model. This model will be used as a basis for the integrated framework.

We present a standalone notation in this section and possible redefine some sets, variables, coefficients or shorthands. A set of trips $\mathcal{T}$ must be serviced by rolling stock units. The set of consecutive trips is denoted $\mathcal{R}$, where every element represents a pair: a trip and it's consecutive trip, which naturally arrives to and departs from the same station. The layout at this station governs which composition changes can be performed at the station between the trips. The set of compositions is defined by $\mathcal{C}$. The set of depots (associated to stations) is defined as $\mathcal{D}$. The set of unit types is defined as $\mathcal{U}$. We define a list of convenient shorthands. Let $\texttt{source}(r)$ and $\texttt{target}(r)$ define the source (first) trip and target (second) trip of connection $r \in \mathcal{R}$, respectively. Let $\texttt{depot}(t)$ define the depot of the arrival station of $t \in \mathcal{T}$. The constants $coup^u_{c,c'}$ and $uncoup^u_{c,c'}$ denote the number of units (of type $u \in \mathcal{U}$) leaving and entering a depot as a result of the transition from composition $c \in \mathcal{C}$ to $c' \in \mathcal{C}$. The constant inv^u_d stipulates how many unit of type $u \in \mathcal{U}$ are initially available at depot $d \in \mathcal{D}$. We define $\mathcal{R}^t_- \subset \mathcal{R}$ to be the set of trip connections that have been coupled at the departure depot of $t \in \mathcal{T}$ ($\texttt{depot}(t)$) by the time the trip departs. Likewise, $\mathcal{R}^t_+ \subset \mathcal{R}$ constitutes the set of connections that have uncoupled at the departure depot of t by the time the trip departs.

Two binary families of variables are introduced. The x^t_c variable has a value of 1 if composition $c \in \mathcal{C}$ is selected for trip $t \in \mathcal{T}$. The variable $z^r_{c,c'}$ is set to 1 if the trip connection $r \in \mathcal{R}$ transitions

from composition $c \in \mathcal{C}$ to $c' \in \mathcal{C}$. The formulation is shown below:

$$\texttt{Minimize:} \quad F(x,z) \tag{7.12}$$

$$\sum_{c \in \mathcal{C}} x_c^t = 1 \qquad \forall t \in \mathcal{T} \tag{7.13}$$

$$x_c^{\texttt{source}(r)} = \sum_{c' \in \mathcal{C}} z_{c,c'}^r \qquad \forall r \in \mathcal{R}, c \in \mathcal{C} \tag{7.14}$$

$$x_{c'}^{\texttt{target}(r)} = \sum_{c \in \mathcal{C}} z_{c,c'}^r \qquad \forall r \in \mathcal{R}, c' \in \mathcal{C} \tag{7.15}$$

$$\begin{aligned} 0 \leq\ & inv^u_{\texttt{depot}(t)} \\ & - \sum_{c \in \mathcal{C}} \sum_{c' \in \mathcal{C}} \sum_{r \in \mathcal{R}^t_-} coup^u_{c,c'} \cdot z^r_{c,c'} \\ & + \sum_{c \in \mathcal{C}} \sum_{c' \in \mathcal{C}} \sum_{r \in \mathcal{R}^t_+} uncoup^u_{c,c'} \cdot z^r_{c,c'} \qquad \forall t \in \mathcal{T} \end{aligned} \tag{7.16, 7.17}$$

$$x_c^t \in \{0,1\} \qquad t \in \mathcal{T}, \forall c \in \mathcal{C} \tag{7.18}$$

$$z_{c,c'}^r \in \{0,1\} \qquad \forall r \in \mathcal{R}, c \in \mathcal{C}, c' \in \mathcal{C} \tag{7.19}$$

The objective is to minimize some function $F(x,z)$ that depends on the values of the decision variables. This is normally a weighted sum of multiple objectives, such as number of trip cancellations, operating cost, demand shortage, shunting costs, end-of-day balance deviations. Constraints (7.13) ensure that every trip is assigned one composition. Note, that the empty composition is included in $\mathcal{C}$ and represents a cancellation. Any composition can be removed from consideration by fixing the x variables to zero, e.g. small platforms may not allow some large compositions. Constraints (7.14) and (7.15) link the set of binary variables. In order to enforce certain rules, a number of the z variables can be removed from consideration by fixing them to zero, e.g. turning trains at stations without depot capacity only allow one transition, namely to the reverse composition. We note, that in the final implementation the model is reduced in order to remove all fixed variables, replacing them with constants instead. Constraints (7.17) ensure that inventory is always non-negative. Finally, the domains of the variables are defined by (7.18)-(7.19).

7.5 Solution Approach

In this section we present two variants of a solution approach for the IRSP. This problem entails solving the SYP, TAP and RSP, which is why we first present solution approaches for these.

A compact formulation for the SYP is described and solved using a commercial MIP solver. Likewise, the TAP can be solved directly, but since it can be cumbersome to generate all pairs of conflicts a priori. We propose an approach that solves a relaxed version of the model, where the conflicts constraints are initially removed and gradually reintroduced via a BAC procedure if violated. The second TAP model is, on the contrary, characterized by an exponential number of parking plans. We describe the column generation model proposed in [8] and show how this can be extended to a full B&P framework through constraint branching.

Finally, we present an integrated framework for solving the IRSP. A general framework is proposed based on separating depot cuts in the rolling stock model. Two variations are proposed based on two similar rolling stock solution approaches. In the first approach, termed RFM, a column generation framework [12] for the RSP is extended to separate depot cuts by solving the TAP. In the second approach, termed RLM, a MIP (Section 7.4.3) is extended to separate depot cuts by solving the SYP. Both frameworks are benchmarked and compared in Section 7.6.

7.5.1 Branch-and-Cut

BAC is a well known technique for efficiently solving large scale MIPs, see e.g. [5]. This approach combines the addition of cutting planes within a B&B framework. More specifically, whenever the relaxation of a node in the branch-and-bound tree is solved to optimality, so-called *separation* routines are performed. Each separation routine attempts to identify valid inequalities, i.e. constraints that are not part of the original formulation, but which are violated by the node's fractional solution. By adding such valid inequalities one hopes to improve the bound by removing infeasible fractional solutions that would otherwise be branched on. The valid inequalities produce a tighter relaxation and often a less fractional solution.

In addition to generating valid inequalities from the problem's full constraint set it is also possible to remove a set of problem constraints, thus obtaining a relaxation, and run a separation routine that identifies any violated, relaxed constraints. This can be particularly useful if there is a large set of constraints, many of which are unlikely to be binding in an optimal solution. See for example, the application of BAC algorithms to the traveling salesman problem and its variants [24], where the exponential number of sub tour elimination constraints can be more efficiently handled via a separation routine.

In addition to directly solving the TAP model (Section 7.4.1), we also compare an alternative BAC algorithm for the TAP. As stated above, we relax the problem by initially removing the LIFO related constraints (Constraints (7.5)), and whenever a violated constraint is found, it is added. Note that if a LIFO violation is found for two events, we add a LIFO constraint for each track in the depot to ensure that the conflict doesn't simply move to one of the other tracks. In this approach we cut on both fractional and integer solutions.

7.5.2 Column Generation

Column generation is an efficient method for solving large scale linear programs (see e.g. [4]) and can, via a B&P algorithm (see e.g. [1]), be adapted to solve large scale mixed integer programs. It is typically used when it is computationally intractable to enumerate all possible variables a priori. In a column generation methodology, the problem is decomposed into a *master* problem and one or more independent *subproblems*. The master problem is solved iteratively considering only a reduced set of the possible variables, while the role of each subproblem is to generate promising variables for the master problem using the dual variables from the optimized master solution, c.f. the pricing step in the Simplex Algorithm. Column generation is an iterative procedure between the master problem and subproblems and continues until no variables with favorable reduced cost are found by any of the subproblems.

As shown in by Freling et al. [8] (and in Section 7.4.2), the problem structure of the TAP is suitable for column generation. The relaxed, restricted master problem is obtained by relaxing the integrality restrictions on the x variables and restricting the set of parking plans in the model. We introduce dual variables π_t for each track $t \in \mathcal{T}$ assignment constraint, and μ_e for each event $e \in \mathcal{E}$ cover constraint. A subproblem can then be identified for each depot track (of distinct length), where the aim is to find an improving parking plan given the optimal solution to the relaxed, restricted master problem. The column generation uses the subproblem to generated a subset of all parking plans; those that are necessary to solve the relaxed formulation. The many variables that will assume a non-basic status in the optimal linear programming are purposely omitted from explicit consideration. As described in [8], the problem of generating favorable reduced cost parking plans can be modeled as a Resource Constrainted Shortest Path (RCSP) problem. We provide a brief overview of this in Section 7.5.2. In particular, we show how the methodology is applied to depot tracks that function as LIFO stacks. This overview is necessary in order to be able to follow the dominance counter example we provide in Section 7.5.2.

Resource Constrained Shortest Path

RCSP problems typically arise in the application of column generation to vehicle routing and crew rostering problems. The problem entails finding a shortest path between two nodes of a network, while adhering to a number of so-called *resource constraints*. The network is a graph representation of the problem at hand, while the resource constraints impose restrictions on the solution to the subproblem.

Freling et al. [8] model the TAP subproblem using an acyclic network. In addition to a source node, $\mathcal{O}$ and sink node $\mathcal{D}$, the node set contains two nodes for each event $e \in \mathcal{E}$, when considering LIFO tracks. The first node indicates that the event is parked on the track and the other indicates the opposite. The nodes for each event constitute a layer, and these layers are sorted by the arrival time of the corresponding events. Arcs are used to connect the nodes of one layer with nodes in the subsequent layer. The source node, $\mathcal{O}$, is connected to nodes in the first layer, while the nodes of the last layer are connected to the sink node, $\mathcal{D}$. Figure 7.3 gives an example of such a network for the TAP with five events. The arcs represent parking choices: whether to park an event on the track or not. All paths originating in $\mathcal{O}$ and terminating in $\mathcal{D}$ represent a plan, which is feasible if the resource constraints are respected. The cost of an arc is the reduced cost contribution of the event. In other words, it is the cost of parking the event c_{et} minus the dual value on the constraint it covers, μ_e. The dual contribution of π_t is a constant applied when track t; for convenience it is included on all arcs connecting the source with the first event nodes. The rectangles in the figure illustrate the layers.

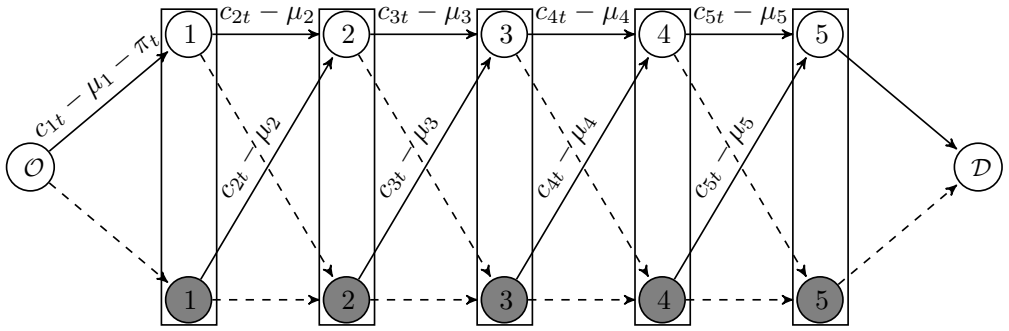


Figure 7.3: An example subproblem network with five events. Every event constitutes two nodes that indicate whether the event is parked or not on the track.

It is clear to see that enumerating all paths in this network would provide all subsets of the considered events. However, not all subsets are feasible. The resource constraints that must be satisfied are the remaining length of the depot track being considered and the minimum departure time of events parked on the track in a partial path. The first ensures track capacity is never violated, while the second prevents conflicting movements from being parked on the track. RCSP problems are typically solved using a label setting algorithm in which the nodes are considered in topological order. A label is associated with a particular node and contains information about the partial path to that point, i.e. the cost and the values of each resource level. New labels are generated by *extending* a label at a given node to each of the node's successor nodes. The new label reflects the label from which it is generated, updated with any changes to resource levels that have occurred in traversing the arc. Dominance strategies are used to restrict the number of labels generated. For an introduction to solving RCSPs, the reader is referred to [17] and [16].

Dominance Rule Counter Example

The proposed solver for the subproblem described in section 7.5.2 is similar to the Dynamic Programming Algorithm described and tested by [8]. A few differences exist, such as the ability

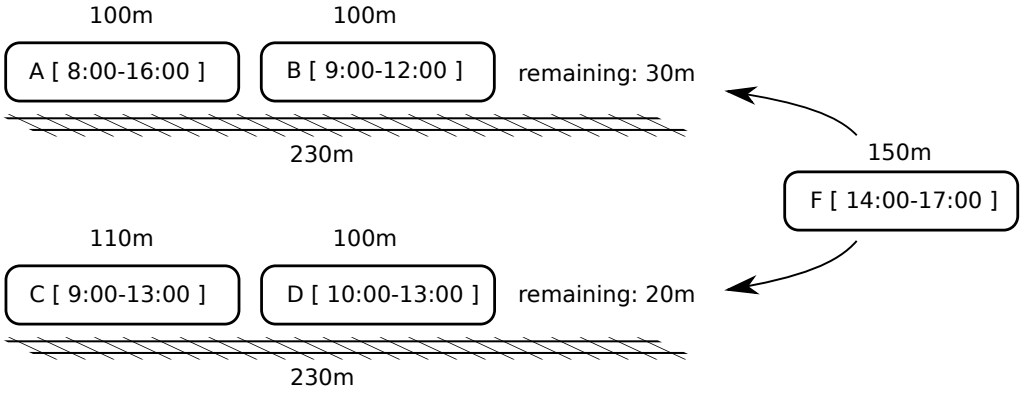


Figure 7.4: Example of two partial parking plans. The boxes represent train units of individual lengths and parking durations. Both plans do not park event E, leaving them in the same state (node in the graph), thus making it possible to check for dominance. The lower plan should not be dominated because the upper plan cannot include event F in contrast to the lower pattern. If F has a negative cost, then a feasible and better pattern is wrongly pruned.

to handle free tracks. We should therefore be able to adopt a specialization of the algorithm as we only consider LIFO tracks. However, we have identified an error in the algorithm.

The proposed domination rule presented by [8] is in fact incorrect. Granted, the overall column generation framework in the paper is heuristic as columns are solely generated in the root node. However, the algorithm does not guarantee optimal solutions for the subproblems because the described dominance rule may prune an optimal solution. We prove this by issuing a counter-example where the algorithm fails to find an optimal solution. We note that the solutions produced by the incorrect algorithm are always feasible, which essentially makes the algorithm a heuristic approach. Freling et al. [8] state the following in section 5.2.2:

> A path $p_i \in P_u$ is dominated by a path $p_j \in P_u$ if all the resource variables of p_i are dominated by those of p_j

This is a standard domination rule that is valid for *regular* resource consumption problems. Parking on tracks is, however, a somewhat temporary resource consumption as it occupies the tracks for a limited time, and blocks units behind it during this period only. The resources in question are the total cost, the remaining length of a track (path) and the "earliest departure time of the blocks in path p, which had not left as of the time of node u", p269. In the case of a LIFO track the last resource is the departure time of the *outmost* rolling stock unit, i.e., the unit which is blocking all the other units on the track.

Our counter example is illustrated in Figure 7.4. In the example a 230 meter track is considered, together with events {A, B, C, D, E, F }, arriving in the listed (alphabetical) order. Assume the costs of the events to be {-2, -2, -1, -2, 10, -10 }, respectively. Consider the case where the solver is testing whether one label (i.e. parking plan) dominates the other one, between events D and F. Assuming that unit E arrives before A, B, C and D have left, the current state of each label remains the same when processing the node, indicating that E will not be parked, see Figure 7.3. At this point the top label and bottom labels have resource consumption vectors (-4, 30, 12:00) and (-3, 20, 13:00). It is clear that by definition the bottom label is dominated because it has a higher cost, less remaining track capacity and the earliest departure time is greater. However, the optimal solution $\{C, D, F\}$ with cost -13 will not be found, as the prefix $\{C, D\}$ has been pruned.

In our preliminary results we identified cases where the non-optimal dominance actually results in non-optimal solutions. As expected, the simpler non-optimal dominance results in faster solution times. The method could therefore be used as a heuristic to speed up the overall convergence in

a B&P framework.

We revise the dominance criteria in order to avoid pruning labels prematurely. In our approach, one label dominates another if, in addition to the remaining length and cost criteria, all units currently parked on the track in the first label have an earlier (or identical) departure time than those of the other label. This would ensure that the top does not dominate the bottom label in Figure 7.4 as unit A is the last of units A, B, C, and D to leave.

Constraint Branching

To ensure optimality of the column generation framework, we utilize the well known constraint branching technique, developed by Ryan and Foster [25], for solving set partitioning problems.

In an optimal, but fractional solution to such a problem there exists two constraints (say c_1 and c_2) such that $\sum_{j \in J(c_1,c_2)} x_j < 1$, where $J(c_1, c_2)$ defines the set of variables that cover (or contribute to) both constraints c_1 and c_2. Two branches are then created by ensuring that either $\sum_{j \in J(c_1,c_2)} x_j \geq 1$ (the *one* branch), or $\sum_{j \in J(c_1,c_2)} x_j \leq 1$ (the *zero* branch). On the *one* branch both constraints must be covered by the same variable, while in the *zero* branch, they must be covered by different variables. For the TAP, we branch on track and event pairs. In other words, on the *one* branch, an event is forced onto a depot track, while on the *zero* branch it is prohibited from using a given depot track. In an optimal, fractional solution, the track and event pair with the closest fractional coverage to 0.5 is selected to branch on.

When branching, all variables that are in the master problem and which do not satisfy the imposed branches are removed, and the subproblems are modified to ensure they return parking plans consistent with the imposed branches. One advantage of this constraint branching approach is that minimal changes to the subproblems are required. Except from removing columns the master problem remains unchanged. When branching, we adopt a depth first strategy in which we successively enforce *one* branches. For set partitioning problems we do not expect the optimality gap to be large. Consequently, such a depth first approach should quickly provide an incumbent of high quality.

7.5.3 Unit Swapping

If not all units can be parked in a solution to the TAP, before rejecting the unit routing solution we attempt to swap interchangeable units. In this paper, we consider two units to be interchangeable if they are of the same unit type. In general, after swapping, the resulting routes should remain feasible, i.e. still satisfy all constraints, examples of which include maintenance or unit mileage restrictions.

A swap results in an exchange of the departure times of the two events in question. Swapping units is only necessary if one or more units could not be parked in the TAP instance. To swap units, we propose a heuristic that iterates over the set of unparked units, $\mathcal{U}$ and, for each, identifies a set of swapping possibilities $\mathcal{S}_u$ by considering the track assignment of each of the parked events in the depot (at the unparked unit's scheduled arrival time) as well as the next units to arrive; i.e. we attempt to swap an unparked unit with an already parked unit (one at the front of a stack) or with one of the next units to arrive. Essentially the heuristic is given an infeasible solution to the SYP in which there is a number of unparked units. It then attempts to insert each of the unparked units into the solution by swapping the departure times with an already feasible unit. The aim being to make minimal changes to guarantee feasibility. The heuristic method is outlined in Algorithm 2.

The algorithm terminates when no unparked events exist, no swapping possibilities remain, or an upper limit on iterations has been reached. We allow one swap per track at each iteration. Swap

Algorithm 2 Unit Swapping Heuristic

```
1: procedure UnitSwap(sol)
2:     iterations ← 0
3:     stop ← false
4:     while unparkedUnits(sol) > 0 and stop=false and iterations < limit do
5:         U ← getUnparkedUnits(sol);
6:         S ← ∅
7:         for u ∈ U do
8:             for t ∈ T do
9:                 S ← S ⋃ createSwapPossibilities(sol, t, u, u_arr)
10:        if |S| > 0 then
11:            rankSwaps(S)
12:            performSwaps(S)
13:            sol ← resolveDepot()
14:        else
15:            stop ← true
16:        iterations = iterations + 1
```

possibilities are ranked such that a unit with fewest swaps is considered first. To ascertain the impact of the chosen swaps, the depot is resolved. The algorithm's sequential nature makes it inherently heuristic.

7.5.4 Integrated Rolling Stock and Depot Framework

In the following we propose a framework that solves the Depot Problem and Rolling Stock Scheduling simultaneously in an integrated loop. The main assumption in this framework is that the depot problem is a feasibility problem, i.e., only the rolling stock schedule contributes to the objective and that no cost stems from the depot problem. Two variants of this framework are proposed: the RFM and the RLM. The distinction lies in the underlying method for solving the RSP and the SYP. In the following we first give a description of the RLM, second we describe in which details the RFM is different. The first is an optimal solution approach for the IRSP. The latter is heuristic as unit routing dictates the matching problem in the SYP.

The RLM is essentially a MIP solved using a BAC methodology. The formulation used is the RSP model described in Section 7.4.3 with a set of additional constraints. The goal of these additional constraints is conceptually to cut all rolling stock schedules for which there exists no valid parking plan. As mentioned, a rolling stock schedule determines when and where units are coupled or uncoupled, i.e. move in and out of the shunting yard. Any given solution can be prohibited by adding a constraint that prohibits one specific solution. The constraint can be improved by only including all movements of a rolling stock solution that only involves the infeasible depot. This can be improved further by only including a minimal subset of movements that are infeasible.

Adopting the formulation and notation in Section 7.4.3, all infeasible solutions (with respects to the shunting yard) are prohibited by adding the following additional constraint:

$$\sum_{(t,c)\in\phi} x_c^t = |\phi| - 1 \quad \forall \phi \in \Psi \tag{7.20}$$

Where Ψ is a full list of all infeasible rolling stock solutions with respect to the shunting yard. Each element in $\phi \in \Psi$ is list of pairs, where each trip is associated to exactly one composition. An infeasible solution is associated to a specific depot (shunting yard), thus in a stronger cut ϕ only specifies a composition for all trips that either enter or leave the infeasible depot. We note, that the cut is still not very strong as similar solutions are not covered by the same constraint,

e.g., the cut for one infeasible solution does not prohibit a very similar solution where one arriving unit is shifted to arrive a little earlier or later.

Many different infeasible rolling stock solutions exist, thus the number of constraints that constitute (7.20) is very large. Enumerating all such constraints is impractical, furthermore, identifying all of them in advance is even more cumbersome. We therefore adopt a BAC framework in the RLM variant, where Constraints (7.20) are initially removed from the formulation and added as they become violated in the B&B tree. Once a rolling stock solution is found, we test whether all depot movements can be resolved. If a feasible shunting plan is found for every depot, then the solution is feasible. Otherwise, a cut is generated for the first depot that has no feasible solution.

The separation routine entails solving the SYP, here we adopt the MIP formulation presented in Section 7.4.1. The rolling stock solution (to be tested) implicitly defines a set of units that move in and out of the shunting yards as a result of coupling or uncoupling. Given the arrivals, departures and the shunting tracks at a specific station, the MIP model can be used to determine whether there exists a feasible shunting yard solution.

For every unparkable rolling stock solution, there exists a cut (Equation (7.20)) that can be inserted to invalidate the infeasible solution. As noted, it is only necessary to include composition assignments that affect the unparkable depot. In order to strengthen the cuts, we try to find a smaller set of events that still cause the solution to be infeasible, thereby making the cut more general and not specific to one or few solutions. We propose an iterative approach. When an infeasible depot is identified, the problem instance is reduced to contain only the first i events. A cut is added for the smallest number of i that is unparkable. Naturally, if the first i events are unparkable, then no subsequent list of events can remedy the solution. Note that in every iteration a new, but reduced, SYP must be solved.

The RFM variant is a similar approach that also adopts a BAC methodology, where cuts are inserted as a results of unparkable rolling stock solutions. The underlying model and approach for solving the RSP is, however, the B&P method described by Haahr et al. [11]. Like the model described in Section 7.4.3, this MIP model also includes binary composition variables that determine which composition is assigned to every trip. Constraints (7.20) are therefore also adopted directly, and here also constitute the cuts that will be found using a separation problem. The rolling stock method of Haahr et al. [11] is an extension of the method by Fioole et al. [6] (Section 7.4.3), as itineraries are identified for every rolling stock unit in contrast to finding a flow of unit types. The extension has a modeling advantage as unit-specific constraints can be modeled, see [11]. Modeling unit-specific constraints is an interesting candidate for future research.

The separation routine for the RFM variant entails solving the TAP problem. In contrast to the separation routine in the RLM, the matching is implicitly determined by the rolling stock itineraries found. The itinerary specifies when a unit enters and leaves a depot, which constitutes a matching. In Section 7.4 we presented different approaches for solving this problem. Either approach can be used interchangeably but in the benchmark we will adopt the direct MIP method.

As before, a cut can be generated if no feasible parking plan exists. However, now there may exist multiple rolling stock route assignments for obtaining the same composition assignment of the train trips. Thus, if no valid parking plan exists for the current rolling stock routes (itineraries), then there may still exist a feasible set of routes. Since enumerating all possible set of routes is intractable, we adopt the method in Section 7.5.3 for identify potential swapping points for the current routes. In an extension, where unit-specific constraints are included, the swapping method can only swap units if the unit-specific constraints are satisfied.

As the rolling stock schedule in the RFM includes unit itineraries (i.e an implicit fixed matching) the separation routine entails solving the TAP. If the TAP instance is infeasible, the unit swapping method tries to find a different feasible set of unit itineraries. If a feasible unit swapping can be obtained, such that units can be parked, we have found a feasible solution. Note, that unless

we apply an exact method for swapping units, the overall is not guaranteed to be optimal as we adopt a heuristic method for swapping units. In Section 7.6 the computational results show the performance of this method, compared to the RLM.

For completeness sake, we note that the RLM can be extended to include rolling stock itineraries. As the name suggests, in contrast to the RFM, the routing is performed last. After finding a feasible rolling stock schedule and corresponding shunting yard plan, the framework could include a step to route the units. Due to unit-specific constraints, there may not exist any feasible routing to the RLM solution, even if one exists to the integrated problem (IRSP). Multiple matchings may exist in the SYP that have to be tested. However, iterating through all possible matchings (across depots) is a trivial task. In conclusion, both variants have strengths and disadvantages. Without modeling unit-specific constraints, the RLM has an advantage. However, when modeling unit-specific constraint, it is less clear which variant is better for solving the IRSP.

7.6 Computational Results

Two types of benchmarks are presented in this section. The first benchmark, in Section 7.6.1, is dedicated to the TAP problem, where we compare two B&P frameworks and two variants of the MIP based approach described in Section 7.5. In the second benchmark we test the proposed integrated framework for solving the IRSP. Two variants are benchmarked and compared, where the TAP constitutes the separation routine in one of them. The results of the integrated framework on DSB S-tog case studies are given in Section 7.6.2. We benchmark and discuss the results obtained by the proposed frameworks. All tests have been performed on a dedicated `Intel(R) Xeon(R)` CPU `X5550` @ 2.67GHz with 24 gigabytes of main memory running `Ubuntu` Linux 14.04. The commercial solvers `Gurobi` 6.0 and `Cplex` 12.6 have been used to experiment with solving the MIP and LP relaxations.

7.6.1 TAP Benchmark

In order to determine the best way of solving the TAP, we benchmark the approaches, described in Section 7.5, on a test set of 11 artificial problem instances. A preliminary study of the available problem instances for DSB S-tog shows that the realistic instances are quickly solved by all methods. They have not been included for this reason. The artificial problem instances constitute a more difficult set of problem instances that vary in the level of infeasibility, the number of events, depot tracks, the number of unit types and other things. Table 7.1 summarizes the problem instances. Note, that the instances have been purposely created with infeasibilities, and that these are in line with the size of the problems DSB S-tog faces. In general these instances have few unit types and consider the maximum number of events that could be encountered in the worst case for DSB S-tog.

We solve the problem instances using four different methods: our column generation approach (*B&P*), two BAC approaches (BAC_1 and BAC_2), and the column generation approach of Freling et al. [8] extended with constraint branching (*FEL*). BAC_1 is a direct solve of the TAP model described in Section 7.4, while BAC_2 also separates violated LIFO cuts, i.e. Constraints (7.5).

The results are given in Table 7.2. An upper bound of 10 minutes is enforced on the solution time. We note, that the objective value here is equal to the number of unparked events in the problem instance. The results show that the BAC_1 outperforms the other methods in general. No added value of separating Constraints (7.5) is observed for the considered problem instances. Infeasibility is expected in these test cases as they have been constructed with this in mind. As such, a subset of Constraints (7.5) must be active in any optimal solution. The generation of these violated inequalities depends on the order in which the integer solutions are found. Given the hardness of the instances, it is understandable that BAC_1 performs better; it has full knowledge of the

Instance	$\|\mathcal{E}\|$	$\|\mathcal{T}\|$	L_{max}	Horizon (s)	Types	Unit Lengths
data0	66	6	300.0	17113	2	[35,70]
data1	69	6	500.0	17785	2	[35,70]
data2	62	5	850.0	21103	2	[42,84]
data3	75	5	850.0	24837	3	[30,60,90]
data4	72	5	700.0	21602	2	[42,84]
data5	59	5	740.0	21602	3	[30,60,90]
data6	79	5	800.0	25202	2	[35,70]
data7	79	6	790.0	25202	3	[35,50,75]
data8	78	7	900.0	24897	1	[42]
data9	101	8	1000.0	24964	2	[42,84]
data10	109	8	400.0	28529	2	[42,84]

Table 7.1: The artificial problem instances. The columns show the instance name, the number of events, the number of depot tracks, the length of the longest track (denoted L_{max}), the planning horizon, the number of unit types, and the unit type lengths.

problem structure, while BAC_2 iteratively cuts away infeasible integer solutions in a potentially very large B&B tree.

The *FEL* approach is slightly faster than the optimal B&P approach, however, it also provides suboptimal solutions. As discussed in Section 7.5.2, this is due to the heuristic nature of the dominance rule presented by Freling et al. [8].

		B&P Framework				BAC_1	BAC_2		*FEL*	
Instance	Z^*	*cols*	n	l	t (s)	t (s)	t (s)	%	t (s)	Z
data0	8	1030	37	18	1.33	3.21	*	12.50	1.30	11
data1	6	1317	41	20	1.74	0.36	*	16.67	1.43	7
data2	7	727	13	6	0.75	0.12	5.05	0.00	0.41	8
data3	8	1181	23	11	1.71	0.11	7.63	0.00	2.38	10
data4	11	834	25	12	1.05	0.14	*	9.09	1.17	14
data5	4	754	29	14	0.98	0.05	1.09	0.00	0.43	8
data6	11	841	17	8	1.18	1.99	*	36.36	0.58	14
data7	8	1740	41	20	2.18	1.76	*	12.50	1.83	9
data8	1	3236	99	49	4.86	0.05	0.36	0.00	3.78	1
data9	3	5125	87	43	17.33	0.42	*	66.67	6.78	3
data10	0	2413	147	73	6.14	0.23	1.29	0.00	5.17	0

Table 7.2: The TAP benchmark results. The first two columns show the problem instance and the optimal objective value, Z^*. For the B&P framework the columns report the number of variables generated (*cols*), the number of nodes in the B&P tree (n), the deepest level of same tree (l) and the time needed to solve the instance (t). For the two BAC approaches the columns show the time and, for BAC_2, the gap from optimality at termination. For *FEL*, the solution time along with the objective value obtained, Z, is reported.

Finally, we benchmark the swapping heuristic. The results are summarized in Table 7.3. We compare the B&P approach with BAC_1. In almost all cases feasibility through swapping can be achieved, and the heuristic nature is seen by the fact that the two approaches can yield different results. The heuristic does not optimize the number of swaps; such a "re-matching" problem could also be performed via a MIP, if running times are fast enough. The integration of the matching and parking in the general SYP is thus motivated by the considered problem instances.

		Column Generation			BAC_1		
Instance	Z^*	Z	$Swaps$	t (s)	Z	$Swaps$	t (s)
data0	8	0	12	7.383	0	19	3.39
data1	6	0	10	20.35	0	11	4.31
data2	7	0	11	5.50	0	16	1.33
data3	8	0	12	15.08	3	15	2.90
data4	11	1	21	126.10	1	16	5.12
data5	4	0	4	3.04	0	5	0.21
data6	11	0	22	18.748	1	18	15.58
data7	8	1	16	145.32	0	18	44.17
data8	1	0	1	8.70	0	1	0.14
data9	3	0	4	42.27	0	4	1.23

Table 7.3: The unit swapping results. For each approach the columns report the new objective (Z), the number of swaps, and the time taken. For any instance, up to 30 swapping iterations are allowed.

Name	Stops	Trips	Trips*	Weekday	Lines
Mon	28 017	4 468	868	Monday	A,B,Bx,C,E,F&H
Fri	28 719	4 558	886	Friday	A,B,Bx,C,E,F&H
Sat	20 474	1 916	590	Saturday	A,B,C&F
Sun	19 919	1 871	574	Sunday	A,B,C&F

Table 7.4: Four timetables operated by DSB S-tog. The columns respectively show the instance names, total number of stops, total number of trips, total number of non-reducible trips (Trips*), weekday, and finally the lines that are running.

7.6.2 Integrated Rolling Stock and Parking

We perform tests on both variants (RFM and RLM) of the integrated framework described in section 7.5.4. Real-life problem instances are provided by the DSB S-tog. They operate a weekly schedule that consists of four daily distinct timetables. The main distinction between the timetables is the difference between weekdays and the weekend, but in addition, Friday and Saturday also include additional night trip services. A time limit of 30 minutes is set. The algorithms are set to terminate when a solution with a proven gap of 1% (relative) is found. The instances are shown in Table 7.4.

In the first benchmark, aimed towards planning in a tactical phase, we do not impose any particular initial position of the rolling stock fleet at the individual shunting yards. In other words, the optimization model can choose freely how to initially park the units in the shunting yards. The results are listed in Table 7.5. For optimizing the rolling stock schedule we have adopted a balanced set of penalties for seat-shortages, driven mileage, end-of-day balance deviations, couplings and uncouplings, see [11]. These initial results demonstrate runtimes which are acceptable for tactical planning purposes.

Surprisingly, all optimal rolling stock schedules are feasible, thus the integrated loop essentially proves unnecessary as no cuts were generated in the BAC framework. At the start of the planning horizon, most units exit the depot and at the end all units enter. In between only a few enter and exit the depots as a results of the rush-hour patterns. With the absence of an initial ordering, the model is always able to order the units such that they can leave initially. Handling the rush-hour patterns is not difficult since the depots are far from being at capacity. Further experiments were performed with varying penalties for coupling and uncoupling units, in effect encouraging more shunting yard activities. These, more artificial problem instances, were however unable to generate depot activities that resulted in infeasibility.

	RFM			RLM		
Instance	Time	#Cuts	Objective	Time	#Cuts	Objective
Fri	44	0	731 839.5	16	0	731 589.8
Mon	355	0	717 798.4	17	0	717 457.6
Sat	5	0	418 791.8	8	0	418 791.8
Sun	5	0	411 940.2	8	0	411 940.2

Table 7.5: Summary of benchmarks performed on all instances without enforcing any particular initial parking. The columns respectively show the instance name, runtime (seconds), number of depot cuts generated and the objective value.

In a short-term or operational planning context, the dispatchers do not have the luxury of deciding the initial rolling stock positions. In contrary, an initial position is given and must be adhered to. In the following we benchmark different initial positions, randomly generated. In our case study only two unit types exist but most shunting yards contain at least three tracks. In effect a random order is not expected to be very limiting.

			Runtime (s)			
Method	Instance	#	Total	Depot	Cuts	Objective
RFM	Fri	10	104	8	8.6	0.10%
	Mon	9	678	13	45.0	0.37%
	Sat	10	54	19	69.4	0.38%
	Sun	10	11	1	9.7	0.09%
RLM	Fri	10	48	24	2.2	0.11%
	Mon	10	50	21	1.9	0.16%
	Sat	10	197	160	35.0	0.57%
	Sun	10	22	7	2.4	0.21%

Table 7.6: Summary of benchmark performed on all instances without enforcing any particular initial parking. The columns respectively show the method, instance name, runtime, number of depot cuts generated and finally the resulting increase in objective relative to the values in Table 7.5. The shown numbers are averages over ten different runs.

.

The results are shown in Table 7.6. A total of ten random initial parkings are generated for every instance and solved using both solution methods. The RLM is able to solve all instances within a few minutes on average. In four cases more than one minute of runtime is observed, and a maximum of approximately 14 minutes. The RFM was unable to find a solution in one case, and ran out of time in another case resulting in a solution with a gap of approximately 1.17%.

The results show that the separation problem in the RLM requires a relatively high portion of runtime. In contrast, the RFM spends more time solving the rolling stock problem. The separation problem of the RFM is expected to be easier to solve as the matching-part of the problem is given. Among the few cases that require most runtime for the RLM, the majority of runtime is spent by the separation problem. A better method for solving the SYP would drastically improve the RLM.

Compared to the previous results in Table 7.5, the increase in objective is relatively small. In the experiments a very large number of rolling stock solutions were identified in the B&B search tree, even more than one hundred in multiple cases, where the majority of the solutions were infeasible. We note that swapping units (Section 7.5.3) is generally not necessary in order to get feasible solutions for the RFM.

7.7 Conclusions

In this paper, we consider the SYP and the RSP, and in addition an exact solution approach for the integrated problem, the IRSP. We present solution approaches to solve the SYP, including a specialization of this problem, the TAP. A column generation approach considered in the literature [6] is shown to be incorrect, in the sense that the so-called subproblem is heuristic due to the dominance rule in the dynamic programming algorithm. We describe how to change the dominance rules in order to maintain overall optimality of the approach. Two variants of the BAC framework for solving the IRSP have been proposed and tested.

We present and benchmark four approaches for solving the TAP. The results, based on our problem instances show no reason to favor a column generation framework over solving a MIP directly. However, given larger problem instances, it is unknown which method scales better. The results also show that separating the violated LIFO constraints in a BAC framework bears no advantage. A direct solve of the MIP achieved the best results, solving each artificial instance within 4 seconds. These instances do represent the largest depot facilities at DSB S-tog. Due to the additional complexity of implementing a column generation approach we deem it appropriate for future research to consider a MIP based solution approach first. The complexity of developing a column generation approach is apparent since even formulating a correct domination criteria can include some pitfalls. We have shown how a previously proposed approach is non-optimal, see [8]. The computational results further showed significant differences between our optimal B&P framework, and the method considered in [8].

Rolling stock scheduling and depot parking planning have traditionally been solved sequentially. Planners at DSB S-tog have identified this sequential approach as problematic, as it is sometimes impossible to find a feasible parking plan for initially found rolling stock schedules. We considered two benchmarks to test our integrated frameworks, the RFM and the RLM, one with and one without an initial parking order. Surprisingly, both frameworks showed no difficulty in solving the integrated problem when no initial parking order is enforced, e.g. in a planning phase. In fact, integration of the RSP and the SYP seems unnecessary. However, given a random parking order, which is a more realistic setting for short-term planning, integration of the problems is justified. The optimal frameworks iterate though multiple rolling stock solutions before finding a high quality solution with a feasible shunting plan.

In future work, it would be interesting to investigate different extension to the RSP or the SYP. Previous work in literature includes some modeling choices with respect to train compositions and marshaling. This addition has been omitted in this work and seems like an obvious choice for future research. Another interesting direction for future research includes train routing at the station and shunting yard. Such considerations could be modeled in the separation problem presented in the proposed framework. Finally, maintenance constraints seem like an obvious extension which possible motivates the RFM in preference to the RLM.

Bibliography

[1] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[2] Ulrich Blasum, Michael R. Bussieck, Winfried Hochstättler, Christoph Moll, Hans-Helmut Scheel, and Thomas Winter. Scheduling trams in the morning. *Mathematical Methods of Operations Research*, 49(1):137–148, 1999.

[3] Nils Boysen, Malte Fliedner, Florian Jaehn, and Erwin Pesch. Shunting yard operations: Theoretical aspects and applications. *European Journal of Operational Research*, 220(1):1–14, 2012.

[4] Jacques Desrosiers, Marco Lübbecke, and Marius M. Solomon. Column generation. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors, *A Primer in Column Generation*, chapter 1, pages 1–32. Springer: New York, 2005.

[5] Jacques Desrosiers and Marco E. Lübbecke. Branch-price-and-cut algorithms, 2010.

[6] Pieter-Jan Fioole, Leo G. Kroon, Gábor Maróti, and Alexander Schrijver. A rolling stock circulation model for combining and splitting of passenger trains. *European Journal of Operational Research*, 174(2):1281–1297, 2006.

[7] Peter Føns. Desicion support for depot planning in the railway industry. Master's thesis, Technical University of Denmark, 2006.

[8] Richard Freling, Ramon M. Lentink, Leo G. Kroon, and Dennis Huisman. Shunting of passenger train units in a railway station. *Transportation Science*, 39(2):pp. 261–272, 2005.

[9] Giorgio Gallo and Federico Di Miele. Dispatching buses in parking depots. *Transportation Science*, 35(3):322–330, June 2001.

[10] Michael Gatto, Jens Maue, Matúš Mihalák, and Peter Widmayer. Shunting for dummies: An introductory algorithmic survey. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos D. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 310–337. Springer Berlin Heidelberg, 2009.

[11] Jørgen T. Haahr, Leo G. Kroon, Lucas P Veelenturf, and Joris C. Wagenaar. Comparing two exact methods for passenger railway rolling stock scheduling. In I.A. Hansen, editor, *Proc. 6th International Conference on Railway Operations Modelling and Analysis (RailTokyo2015)*. 2015.

[12] Jørgen Thorlund Haahr, Richard Martin Lusby, Jesper Larsen, and David Pisinger. *A Branch-and-Price Framework for Railway Rolling Stock Rescheduling During Disruptions*. DTU Management Engineering, 2014.

[13] Réne. Haijema, C.W. Duin, and Nico M. van Dijk. *Train Shunting: A Practical Heuristic Inspired by Dynamic Programming*, pages 437–475. John Wiley & Sons, Inc., 2006.

[14] Mohamed Hamdouni, Guy Desaulniers, Odile Marcotte, François Soumis, and Marianne van Putten. Dispatching buses in a depot using block patterns. *Transportation Science*, 40(3):364–377, 2006.

[15] Mohamed Hamdouni, Guy Desaulniers, and François Soumis. Parking buses in a depot using block patterns: A benders decomposition approach for minimizing type mismatches. *Comput. Oper. Res.*, 34(11):3362–3379, November 2007.

[16] Stefan Irnich. Resource extension functions: properties, inversion, and generalization to segments. *OR Spectrum*, 30(1):113–148, 2008.

[17] Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In Guy Desaulniers, Jacques Desrosiers, and MariusM. Solomon, editors, *Column Generation*, pages 33–65. Springer US, 2005.

[18] Per Munk Jacobsen and David Pisinger. Train shunting at a workshop area. *Flexible Services and Manufacturing Journal*, 23(2):156–180, 2011.

[19] Leo Kroon and Dennis Huisman. Algorithmic support for railway disruption management. In Jo A.E.E. Nunen, Paul Huijbregts, and Piet Rietveld, editors, *Transitions Towards Sustainable Mobility*, pages 193–210. Springer Berlin Heidelberg, 2011.

[20] Leo G. Kroon, Ramon M. Lentink, and Alexander Schrijver. Shunting of passenger train units: An integrated approach. *Transportation Science*, 42(4):436–449, 2008.

[21] Ramon M. Lentink, Pieter-Jan Fioole, Leo G. Kroon, and Cor van't Woudt. *Applying Operations Research Techniques to Planning of Train Shunting*, pages 415–436. John Wiley & Sons, Inc., 2006.

[22] Lars Kjær Nielsen. *Rolling Stock Rescheduling in Passenger Railways*. PhD thesis, Erasmus University Rotterdam, 2011.

[23] Lars Kjær Nielsen, Leo Kroon, and Gábor Maróti. A rolling horizon approach for disruption management of railway rolling stock. *European Journal of Operational Research*, 220(2):496–509, 2012.

[24] Manfred Padberg, , and Giovanni. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1–7, March 1987.

[25] David M. Ryan and Brian A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*, pages 269–280. North-Holland, 1981.

[26] Alexander Schrijver. Minimum circulation of railway stock. *CWI QUARTERLY*, 6:205–217, 1993.

[27] Gabriele Di Stefano and Magnus Love Koči. A graph theoretical approach to the shunting problem. *Electronic Notes in Theoretical Computer Science*, 92(0):16 – 33, 2004. Proceedings of ATMOS Workshop 2003.

[28] Wout van Wezel and Jan Riezebos. Case study: Advanced decision support for train shunting scheduling. In Jan C. Fransoo, Toni Waefler, and John R. Wilson, editors, *Behavioral Operations in Planning and Scheduling*, pages 413–430. Springer Berlin Heidelberg, 2011.

[29] Oliver Weide, David Ryan, and Matthias Ehrgott. An iterative approach to robust and integrated aircraft routing and crew scheduling. *Computers & Operations Research*, 37(5):833 – 844, 2010. Disruption Management.

[30] Thomas Winter and Uwe T. Zimmermann. Real-time dispatch of trams in storage yards. *Annals of Operations Research*, 96:287–315, 2000.

Chapter 8

Exact Methods for Solving the Train Departure Matching Problem

Jørgen Thorlund Haahr* Simon Henry Bull*

*Department of Management Engineering, Technical University of Denmark,
Produktionstorvet, Building 426, DK-2800 Kgs. Lyngby, Denmark
jhaa@dtu.dk, simbu@dtu.dk

[1] **Abstract** In this paper we consider the train departure matching problem which is an important subproblem of the *Rolling Stock Unit Management on Railway Sites* problem introduced in the ROADEF/-EURO Challenge 2014. The subproblem entails matching arriving train units to scheduled departing trains at a railway site while respecting multiple physical and operational constraints. We formally define that subproblem, prove its NP-hardness, and present two exact method approaches for solving the problem. First, we present a compact *Mixed Integer Program* formulation which we solve using a MIP solver. Second, we present a formulation with an exponential number of variables which we solve using column generation. Our results show that both approaches have difficulties solving the ROADEF problem instances to optimality. Due to the complexity of solving the instances for this problem we have developed a heuristic based on both approaches. The column generation approach is able to generate good quality solutions within a few minutes in a heuristic setting.

8.1 Introduction

Many railway planning problems have been studied in the literature for the last two decades. These range from long term high and level planning problems, such as line planing, to detailed and short term rolling stock and crew scheduling problems. At train stations, planning problems include platform assignment, routing, and shunting. These problems often have direct or indirect dependencies but due to the high complexity, or company organizational structure, they are often solved in isolation and in sequential order. The *ROADEF/EURO Challenge 2014* presents a problem where the goal is to adopt a holistic approach to the planning problems at railway stations. The problem combines several planning aspects that must be handled between arrivals and departures at terminal stations such as matching available trains to departures, routing trains

[1]Submitted to a special issue of *Annals of Operations Research*, January 2015

in the station infrastructure (without any two trains occupying the same infrastructure without sufficient time between them), determining whether and when to perform maintenance, and when and where to do the couplings and decouplings of train convoys.

In this paper we present our contribution for the Departure Matching Problem (DMP) in this challenge. The DMP can be considered a pure subproblem of the problem presented in the *ROADEF/EURO Challenge 2014* document [11], which we will refer to as the Rolling Stock Unit Management on Railway Sites (RSUM) problem.

The RSUM problem entails many different aspects but the performance of any solution approach will be greatly affected by how trains are matched to departures. The considered subproblem (DMP) is the problem of finding a good and feasible matching of trains to departures while respecting train compatibility and maintenance constraints. In contrast to the RSUM problem the DMP does not consider how to route, couple and de-couple train units in the station. For the purpose of this paper, we will assume that routing is done in a subsequent step. Importantly, many routing decisions are motivated by the matching of trains to departures; whether an arriving train should visit a maintenance facility or be parked in a yard depends on the train's subsequent departure.

In this paper we propose and benchmark two distinct optimal solution methods for solving the DMP. We will consider finding solutions that are either optimal or proven to be some percentage from the optimum. The proposed solution methods are however flexible and can be adjusted to find solutions in a heuristic manner. We investigate the potential of the methods in a heuristic setting.

8.1.1 Our Contributions

We present a definition for the DMP; a distinct, self contained subproblem of the RSUM. We prove the DMP to be NP-hard in the strong sense. Two optimization-based approaches for solving instances of the DMP are proposed. Firstly, we introduce a Mixed Integer Program (MIP) mathematical model and present results produced using a commercial MIP solver. Secondly, we propose an alternative but equivalent MIP mathematical model that is solved using column generation. Due to time limitations and the hardness of the instances we only solve the pricing problem at the root node. The results are presented in the benchmark section.

8.2 Problem Definition

A full solution to the RSUM problem requires routing trains in the station infrastructure, respecting routing restrictions, headways, capacities, and many other constraints. Solutions are ranked using a weighted sum of multiple objectives such as making preferred matchings, allocating arrivals and departures to preferred platforms, avoiding unnecessary and platform dwell times. Solving the entire problem as a single optimization problem is intractable considering the given strict run-time requirements of the competition. The RSUM problem can be decomposed into several subproblems. The first subproblem, namely the DMP, is the scope of this paper.

A solution to the matching subproblem (DMP) can be used to build a solution to the overall RSUM problem, taking into account its other constraints and objectives. However the fixed matching provided from the DMP may lead to suboptimal solutions to the entire problem. Our approach for solving instances of the RSUM is to first solve a DMP instance, and then use an optimal approach to assigning platforms, before heuristically finding train routes. Our solution method is described in [3]. The DMP contains components of the RSUM that are weakly linked to following subproblems in order to improve the tractability of the RSUM.

The DMP is a matching problem between arrivals and departures at some terminal station, with complicating dependencies between potential matches. Given a fixed planning horizon at the station, there is a set $\mathcal{A}$ of trains arriving and a set of pre-specified departures $\mathcal{D}$ that must be assigned some compatible train unit. Every arrival $a \in \mathcal{A}$ has an arrival time $arrTime_a$, and every departure $d \in \mathcal{D}$ has a departure time $depTime_d$. One arrival corresponds to a single train unit, and if several trains arrive together as a convoy then they are represented by multiple arrivals with the same arrival time. Similarly one departure exists for every train that departs in a convoy sharing the same departure time. A set $\mathcal{I}$ of initial trains may reside in the station infrastructure at the start of the planning horizon. These are all available from the start of the planning horizon h_0. The total set of trains is denoted by $\mathcal{T} = \mathcal{I} \cup \mathcal{A}$. We define the availability time of a train $t \in \mathcal{T}$ as $startTime_t$ where $startTime_t = arrTime_t$ if t is part of an arrival and $startTime_t = h_0$ if t is an initial train (i.e. in $\mathcal{I}$). Note, that h_0 denote the start of the planning horizon.

Some departures are the beginning of a tour that returns the train units to the terminal station as arrivals within the same planning horizon. If an arrival is linked to an earlier departure we call it a *linked* arrival. Likewise we call the earlier departure a *linked* departure. In order to avoid confusion, we note that the "linked" concept does *not* refer to physical train units that are coupled together to form a convoy. A linked arrival states that the arriving train is the same train that was assigned to the corresponding (earlier) linked departure. In contrast to a non-linked arrival, this means that the arriving train is the same physical train which was assigned to the linked departure. If the linked departure is canceled, then a new replacement train arrives instead. The linking between arrivals and departures is important because it means that the properties of (linked) arrivals are not known, without knowing what trains (if any) are matched to those arrivals' linked departures. We define the set $\mathcal{L} \subseteq \mathcal{A}$ as the set of arrivals which have an earlier linked departure, and define $\sigma(a)$ as the linked departure of the arrival train $a \in \mathcal{L}$.

Every train $t \in \mathcal{T}$ belongs to some category $cat_t \in \mathcal{C}$ that defines common characteristics such as train length, capacity and maintenance durations. Two trains of the same category are considered interchangeable when assigning trains to departures – with one exception. All trains in the system are subject to two types of maintenance constraints: Distance Before Maintenance (DBM) and Time Before Maintenance (TBM). Each train t has some initial *remaining DBM* ($remDBM_t$) and *remaining TBM* ($remTBM_t$). Every departure $d \in \mathcal{D}$ has a *required DBM* ($remDBM_d$) and a *required TBM* ($remTBM_d$) to perform the round-trip. If a train is to be matched to departure d, then it must have sufficient DBM and TBM.

A train may visit a maintenance facility at the station between arriving and departing, which in the RSUM problem takes a fixed amount of time and resets both the DBM and TBM to their maximum value for the train, and we include the decision of whether or not to perform maintenance in the DMP. The constants $maxDBM_t$ and $maxTBM_t$ indicate the level of DBM and TBM that is obtained if a maintenance operation is performed on train t. For trains of the same category ($i \in \mathcal{C}$) the constants have identical values, and we can therefore use the notation $maxDBM_i$ and $maxTBM_i$ without ambiguity. Due to a limited amount of manpower at the maintenance facilities the total number of maintenance operations per day is limited to a constant of $maxMaint \in \mathbf{Z}^+$. The imposed limit means that certain combinations of matches can not all be made: if there are $n > maxMaint$ otherwise independent matches that would all occur on the same day and all require maintenance, at most $maxMaint$ of them can be made. The remaining $(n - maxMaint)$ trains could not be maintained and would therefore not have sufficient DBM and TBM to be matched to the $(n - maxMaint)$ departures.

For those arrivals that have a linked departure, category, DBM and TBM are dependent on any matching made to that linked departure. An arrival $a \in \mathcal{L}$ is linked to a previous departure $d \in \mathcal{D}$, i.e., the train $t \in \mathcal{T}$ assigned to departure d is the same train arriving later in a. The train in a therefore inherits the remaining DBM and TBM and category of train t. In the case where no train is assigned to d then another new train arrives with its own specified remaining DBM, TBM, and category.

Every train $t \in \mathcal{T}$ can only be matched with a limited set of departures. We define $CompDep(t)$ as the set of departures that are compatible with t. Likewise we define $CompTr(d)$ as the set of trains that are compatible with departure $d \in \mathcal{D}$. These two sets are considered as parameters to the problem, and it is up to the end-user to specify which factors to consider. These factors could include for example a minimum routing time, a maximum time between arrival and departure, or a maximum number of potential matchings for any given train. For the sake of simplicity, we define only a few simple rules for possible matchings. Given a departure $d \in \mathcal{D}$ and a non-linked arrival train or initial train $t \in \mathcal{T}$ a matching is possible if the following conditions are satisfied:

$$cat_t \in compCatDep_d$$
$$startTime_t + maintenance_t > depTime_d$$
$$\max\{remDBM_t, allowDBM_t \cdot maxDBM_t\} \geq reqDBM_d$$
$$\max\{remTBM_t, allowTBM_t \cdot maxTBM_t\} \geq reqTBM_d$$

where the binary parameters $allowDBM_t$ and $allowTBM_t$ indicate whether DBM and TBM are allowed to be performed. In some cases there is only time for one of the two operations but not both. The constant $maintenance_t$ indicates the time needed to perform the necessary maintenance operations for train t, or zero if no maintenance is required.

Given a departure $d \in \mathcal{D}$ and a linked arrival $t \in \mathcal{L}$ it is harder to limit the options beforehand. In the preprocessing it can only be restricted by $startTime_t > depTime_d$ as the category, remaining DBM and remaining TBM of the linked arrival t are unknown.

In practice it is in some cases expected that certain arrivals are matched with specific departures. If such a match is successful we call it a *train reuse*, otherwise it is a missed train reuse. We denote $\mathcal{U}$ as the set of train uses, where tr_u and dep_u define the train $t \in \mathcal{T}$ and departure $d \in \mathcal{D}$ for a reuse $u \in \mathcal{U}$.

Definition 8.1 (The Departure Matching Problem Definition)**.** *We define the DMP as the problem of finding a feasible matching between trains and departures that respects the departure maintenance requirements, the departure train category compatibility, the time required to perform the needed maintenance, and the total number of maintenance operations per day. The objective of the DMP is to minimize the number of uncovered departures and to maximize the number of train reuses. In every instance of the problem there is a fixed penalty for every missed train reuse and a fixed penalty for every uncovered departure.*

Figure 8.1 shows a small example with three arrival trains (A_1, A_2, A_3) and two departure trains (D_1, D_2), considering three different train categories having no maintenance requirements or other restrictions. In this example departure D_1 and arrival A_3 form a *linked arrival* pair: whatever is matched to departure D_1 returns as arrival A_3. Arrival train A_3 is in $\mathcal{L}$, and its linked departure is D_1; that is, $\sigma(A_3) = D_1$. In the figure, arrival A_3 has category $c3$ marked as its *replacement train* category. That is, it *only* has category c_3 if no match is made to departure D_1, but if instead some match is made, the category of A_3 is inherited from that match. The match between arrival A_3 and departure D_2 is not independent of other matchings made; it is only feasible if a match is also made between arrival A_2 and departure D_1. If D_1 is unmatched, arrival A_3 will have the category of its replacement train, incompatible with departure D_2. Similarly, if D_1 is matched to arrival A_1 then arrival A_3 will inherit the category of arrival A_1, also incompatible with departure D_2.

8.3 Related Problems

The RSUM as defined for the ROADEF competition is based on real station infrastructure and problems, with certain simplifications to make it appropriate for the competition, such as simplifications of the switching and yard infrastructure. Real train stations face similar problems, though

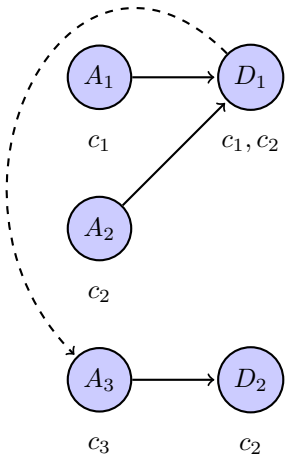


Figure 8.1: An illustration of the matching problem showing potential matches between three arrival trains (A_1, A_2, A_3) and two departures (D_1, D_2). Each arrival has a train category (c_1, c_2, c_3) and each departure has one or more acceptable train categories. Departure D_1 and arrival A_3 are linked; arrival A_3 has replacement category c_3 only if departure D_1 is unmatched, but instead has category c_1 or c_2 depending on which arrival is matched to D_1. The only matching of cardinality 2 is $\{(A_2, D_1), (A_3, D_2)\}$; matching A_1 with D_1 precludes matching A_3 with D_2 as the category of arrival A_1 is incompatible with departure D_2.

those problems differ in specific details. A matching problem similar to the DMP subproblem (that we have identified) could also exist as a subproblem at stations, though it may not necessarily be treated as a self-contained subproblem.

If the linking between some departures and later arrivals is ignored or not present, and performing maintenance is ignored, then whether or not a match is possible would be pre-determinable. If the problem was just that of minimizing the number of uncovered departures, then it would be a relatively simple maximal bipartite matching problem, solvable in polynomial time [4]. If minimizing a weighted sum of uncovered departures and missed reuses, the problem could be formulated as an assignment problem, also solvable in polynomial time [9].

The presence of linked arrivals inherently changes the structure of the problem. The matchings themselves are not independent because whether or not some train can be matched to some departure can depend on what *other* train is matched to some *other* departure. Similarly, the ability to perform maintenance changes which matches are possible, and the restriction of the maximum number of maintenance operations per day makes matches non-independent. The DMP combines the matching with components of the RSUM that we have identified as being closely related to and significantly interacting with the matching, without including so many aspects of the RSUM to make the problem intractable. For example, in the RSUM problem the restriction of the maximum number of maintenance operations per day means that some sets of potential matches can not all be included, and maintenance decisions should be included with the matching subproblem. If however there was no restriction on the maximum number of maintenance operations per day then perhaps a subproblem that ignores maintenance could be sufficient to provide feasible or optimal solutions, relying on it always being possible to perform maintenance if necessary. For some other similar station arrival problem, a similar but distinct problem to the DMP might instead be identified as a subproblem.

Freling et al. identify the subproblem of matching departures and arrivals as part of a shunting

$$\max \sum_{i\in\{1,2,3\}} p_1 x_{i1} + p_2 x_{i2} + p_3 x_{i3} + p_4 x_{i4}$$

$$w_1 x_{11} + w_2 x_{12} + w_3 x_{13} + w_4 x_{14} \leq c_1$$
$$w_1 x_{21} + w_2 x_{22} + w_3 x_{23} + w_4 x_{24} \leq c_2$$
$$w_1 x_{31} + w_2 x_{32} + w_3 x_{33} + w_4 x_{34} \leq c_3$$

$$\sum_{i\in\{1,2,3\}} x_{ij} \leq 1 \quad \forall j \in \{1,2,3,4\}$$

$$x_{ij} \in \{0,1\}$$

Figure 8.2: Multiple Knapsack Problem instance with 4 (coloured) items and three knapsacks.

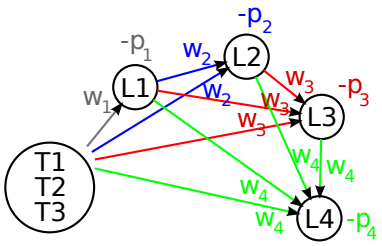


Figure 8.3: The constructed Train Matching Problem instance from the Multiple Knapsack Problem instance shown in Figure 8.2. Each train ($T1, T2, T3$) corresponds to a single knapsack, and each linked departure ($L1, L2, L3, L4$) corresponds to one item. For the sake of clarity, initial trains have been grouped together since the graph is identical for each train.

problem [2]. The authors formulate a matching subproblem that considers the unattractiveness of producing matches that require breaking up trains into units matching different departures. In contrast, in the DMP we do not include any cost or penalty for matches which require coupling or decoupling. The authors do not describe anything similar to linked arrivals or maintenance decisions and daily restrictions, which are the features of our problem that make matches inter-dependent.

Kroon et al. identify a matching subproblem as part of a larger station shunting problem [8]. However there is no analogue to the linked arrivals of the DMP. The authors do not solve the matching problem in isolation but as part of a larger formulation that includes shunting features that are not part of the DMP or even necessarily part of the RSUM.

In the shunting literature there are many problems that share similarities with the RSUM problem and potentially have a subproblem that is very similar to the DMP. However shunting is not necessarily an important component of the ROADEF challenge because the station infrastructure has large and simplified "yard" resources. These are abstractions with only a maximum capacity, but train units can be parked in or removed from the yards without considering their ordering.

8.4 NP-hardness

It is relatively simple to prove that the DMP is NP-hard by reduction from the Knapsack Problem. However, since the classic version of the Knapsack Problem is NP-hard in the weak sense, we prove the same by reduction from the 0-1 Multiple Knapsack Problem (MKP). We adopt the definition of the MKP of [7]:

Definition 8.2. *Given a list of items to pack, each with a profit p_i and weight w_i, and one or more knapsacks of capacity c_j. The 0-1 Multiple Knapsack Problem is the problem of choosing a list of items for every knapsack such that the profit of the selected items is maximized while respecting the weight-restriction of every knapsack.*

Note that the 0-1 variant only allows each item to be packed at most once. We show NP-completeness by reduction from a MKP instance to a DMP instance with a simple mapping from knapsacks to trains and items to departures. All physical train units have a DBM constraint that must be respected, this will be the map to the capacity of a knapsack. All departures assigned to a train consume some level of DBM and a profit/penalty is achieved/given, this corresponds to putting an item into the knapsack. The weight corresponds to the required level of DBM.

An illustrative example of the transformation is shown in Figure 8.2. The knapsack example contains four items and three knapsacks with individual knapsack capacities where every item can be packed at most once. The resulting DMP instance is shown in Figure 8.3.

Consider a MKP instance with N items and M knapsacks, let the weights and profits respectively be defined as w_i and p_i for every item i, and finally the capacities as c_j for every knapsack j.

We construct a DMP instance with M initial (or arrival) trains and N linked departures. Each individual train has remaining DBM corresponding to one of the capacities c_j of a knapsack. We set the remaining TBM to some high value such that this constraint is never binding. The departure and arrival times are set such that any of the four trains could be matched to any of the four linked departures, provided that it has sufficient remaining DBM. Any train can also visit any subset of the departures. The cost of matching every respectable (linked) departure is set to the negative profit of the item corresponding to each departure in the MKP instance. The cost of cancelling a departure is set to zero, and w.l.o.g. no maintenance is allowed. We can ensure that no maintenance will be performed by enforcing zero allowed maintenance operations, or by setting a high cost on maintenance, or by setting the replenished value to zero after maintenance. No train-reuse costs are defined. The initial trains are of the same single category, and all departures (of the linked departures) are compatible with only that train type. The replacement train of the linked arrivals have a different (incompatible) train type, and therefore do not influence the solution or quality in any way. Thus it is only possible for the initial trains to be matched to any of the linked departures. The optimal solution of the DMP instance will therefore only match the linked departures with the physical trains starting as the specified initial trains. Further, the matching-sequences are restricted by the given capacities of the MKP instance, where every matching/item consumes the specified knapsack weights. Due to the departure constraint, every linked departure can be matched at most once, thus every item is at most assigned to one knapsack. If any train is matched to linked departure i, $-p_i$ is added to the objective and inflicts a consumption ($reqDBM_i$) of w_i to the DBM of the train. As no costs, but the profits of the MKP instances are present, the optimal solution value will correspond to the optimal value of the MKP instance, only with a negative sign. Alternatively, in order to more closely follow the DMP objective function we adopt in later sections, the profit can be reformulated in terms of minimizing the cost of cancellations. A item of the MKP instance is only picked in a particular knapsack iff the corresponding linked departure is matched in the DMP instance to a particular train (initially, or as a returning linked arrival of that train). In conclusion, by transforming (in polynomial time and resources) an instance of the MKP is solved by solving the constructed DMP instance. The transformation graph is polynomial in the number of vertices and edges. The number of vertices is $1 + M$. The graph is acyclic and every node only connects forward in time (with respect to arrival and departure times) which results in a total of $\sum_{k=1}^{N} k = 1/2(N^2 + N)$ edges.

Theorem 8.1. *The DMP is NP-hard*

Proof. It is straightforward to verify whether a solution to the DMP is feasible or not. This is verified by checking that the DBM and TBM constraints are valid, that the train types are compatible, and that the matched arrival and departures times are respected by the train units.

Likewise, it is simple to calculate the objective cost of any given feasible solution. The number of operations and resources used to construct the DMP instance is polynomial. The number of vertices and edges created are polynomial in the MKP instance size. By reduction from the MKP, as described in the example, we have argued that there is a one-to-one correspondence between the solution of the constructed DMP instance and the MKP instance. Any feasible train matching solution to the DMP instance is a feasible item selection for the MKP instance, and vice versa. □

8.5 Mixed Integer Program Model

In this section we present a MIP mathematical model for the DMP. The size of the proposed MIP model for the DMP is polynomial in the number of input trains and departures, and can be solved using a commercial solver.

The model contains six types of variables. A set of binary variables m_t^d determine whether train $t \in \mathcal{T}$ is matched to departure $d \in \mathcal{D}$. Matches that are not present in the compatibility set ($CompDep(t)$ or $CompTr(d)$) are omitted or fixed to zero. We also introduce a set of binary variables cat_t^i that indicate if train $t \in \mathcal{T}$ is of category $i \in \mathcal{C}$. For all initial trains and non-linked arrival trains the category is known and the corresponding variable can be fixed (or omitted). The continuous variables dbm_t and tbm_t determine the DBM and TBM of train $t \in \mathcal{T}$ at the time before departure, which is the initially available DBM/TBM or $maxDBM_t/maxTBM_t$ if maintenance is performed on the train. Finally we introduce the binary variables f_t^d and g_t^d that determine whether a train $t \in \mathcal{T}$ matched to $d \in \mathcal{D}$ is being maintained on DBM or TBM. In the following we will assume that maintenance of DBM and TBM can only be done on one specific known day, for a particular matching. We introduce a binary parameter $\omega_{t,day}^d$ that is 1 if a match between t and d would perform maintenance on day (if it performs it at all), and 0 otherwise. The objective is formulated as a minimization of the number of unmatched departures and the cost of missed train reuse:

$$\begin{aligned}\min \sum_{d \in D} & cancellationCost_d \cdot c_d \\ & - \sum_{u \in \mathcal{U}} reuseCost \cdot m_{tr_u}^{dep_u} \\ & + reuseCost \cdot |\mathcal{U}|\end{aligned}$$

We minimize the number of cancelled trains and maximize the number of reuses. Note that the final term is constant and can be left out. The constraints of the model are the following:

$$\sum_{t \in \mathcal{T}} m_t^d \geq 1 - c_d \qquad d \in \mathcal{D} \qquad (8.1)$$

$$\sum_{t \in \mathcal{T}} m_t^d \leq 1 \qquad d \in \mathcal{D} \qquad (8.2)$$

$$\sum_{d \in \mathcal{D}} m_t^d \leq 1 \qquad t \in \mathcal{T} \qquad (8.3)$$

$$\sum_{t \in \mathcal{T}} \sum_{d \in \mathcal{D}} \left(\omega_{t,day}^d f_t^d + \omega_{t,day}^d g_t^d\right) \leq maxMaint \qquad day \in \mathcal{H} \qquad (8.4)$$

$$f_t^d + g_t^d \leq 2m_t^d \qquad t \in \mathcal{T}, d \in \mathcal{D} \qquad (8.5)$$

Constraints (8.1) ensure that every departure is assigned (to some train), unless there is a cancellation. Constraints (8.2) ensure that at most one train is assigned to every departure. Constraints (8.3) ensure that each train is assigned at most once. Constraints (8.4) ensure that

the total number of maintenance operations (every day) is respected. Constraints (8.5) prohibit maintenance usage on a match if that match is not made.

$$\sum_{i \in \mathcal{C}} cat_t^i = 1 \qquad t \in T \tag{8.6}$$

$$m_t^d \leq \sum_{i \in compCatDep_d} cat_t^i \qquad t \in \mathcal{T}, d \in \mathcal{D} \tag{8.7}$$

$$cat_t^i \geq cat_{t'}^i + m_{t'}^{\sigma(t)} - 1 \qquad t \in \mathcal{T}, t' \in \mathcal{T} \setminus \{t\}, i \in \mathcal{C} \tag{8.8}$$

$$cat_t^{i_t} \geq c_{\sigma(t)} \qquad t \in \mathcal{T} \tag{8.9}$$

Constraints (8.6) ensure that every train is assigned exactly one category. Again, the constraint for initial trains and non-linked arrival trains can be omitted, if the correct value is fixed. Constraints (8.7) ensure that trains cannot be assigned to departures where the category is not compatible. Constraints (8.8) ensure that if train t' is matched to the linked departure $\sigma(t)$ of train t, then train t inherits the category i of train t'. Finally, constraints (8.9) ensure that, if the linked departure $\sigma(t)$ of train t is not covered, then train t has the category i_t of its replacement train.

$$\begin{aligned} dbm_t \leq & \sum_{t' \in \mathcal{T} \setminus \{t\}} \left(dbm_{t'}^{\sigma(t)} - reqDBM_{\sigma(t)} \right) \cdot m_{t'}^{\sigma(t)} \\ & + c_{\sigma(t)} \cdot remDBM_t \\ & + \sum_{d \in \mathcal{D}} f_t^d \cdot M \end{aligned} \qquad t \in \mathcal{T} \tag{8.10}$$

$$dbm_t \leq \sum_{i \in \mathcal{C}} maxDBM_i \cdot cat_t^i \qquad t \in \mathcal{T} \tag{8.11}$$

$$0 \leq dbm_t - m_t^d \cdot reqDBM_d \qquad t \in \mathcal{T}, d \in \mathcal{D} \tag{8.12}$$

Constraints (8.10) ensure that the available DBM for a train t is correct. The right hand side consists of three terms. The first term counts the contribution from trains that are linked to t. If the train is cancelled the term sum is zero, and the second term will contribute with $remDBM_t$ which is the *new* train inserted in case of a cancellation of departure $\sigma(t)$. The third term makes the constraint non-binding if maintenance is performed; the constant M is a big number that is no less than the maximum of $maxDBM_i$ for all $i \in \mathcal{C}$. For the sake of clarity the constraints have been presented using a non-linearity in the first term. In order to maintain a linear model we replace the constraints with linear constraints described in section 8.5.1. Since the train category i of a train t might be unknown we further constraint the DBM of the train to respect the $maxDBM_i$ in Constraints (8.11). Constraints (8.12) make sure that enough DBM is available for a matching. For all matchings that are not active the constraint is just requiring dbm_t to be non-negative.

We will not present the corresponding constraints for TBM since they are analogous to Constraints (8.10), (8.11) and (8.12).

8.5.1 Reformulating the Nonlinear Constraints

The first term on the right hand side of Constraints (8.10) is non-linear and can be remodeled as a linear constraint by adding one more group of continuous variables and additional constraints. We introduce one new auxiliary variable κ_t^{dbm} for each train t that measures the DBM contribution of the train linked to departure $\sigma(t)$. We replace Constraints (8.10) with the following set of

constraints:

$$dbm_t \leq \kappa_t^{dbm} + c_{\sigma(t)} \cdot remDBM_t + \sum_{d \in \mathcal{D}} f_t^d \cdot M \qquad t \in \mathcal{T} \tag{8.13}$$

$$\kappa_t^{dbm} \leq dbm_{t'} - m_{t'}^{\sigma(t)} \cdot reqDBM_{\sigma(t)} \qquad t \in T, t' \in \mathcal{T} \setminus \{t\} \tag{8.14}$$
$$+ \sum_{d \in \mathcal{D} \setminus \sigma(t)} m_{t'}^d \cdot M + \sum_{t'' \in \mathcal{T} \setminus t'} m_{t''}^{\sigma(t)} \cdot M$$

$$\kappa_t^{dbm} \leq M \cdot (1 - c_{\sigma(t)}) \qquad t \in T \tag{8.15}$$

The three terms of Constraints (8.13) are analogous to the three terms of Constraints (8.10), except the first non-linear term is replaced with the new contribution term κ_t^{dbm}. Constraints (8.14) have four terms defining an upper bound on the linked contribution for train t. The first two terms are relevant if train t' is matched to the linked departure $\sigma(t)$ and ensure the contribution is no greater than the difference between the DBM for t' and the required DBM for departure $\sigma(t)$. The third term loosens any bound on κ_t^{dbm} related to t' if t' is matched to some departure other than $\sigma(t)$. The fourth term loosens the bound on κ_t^{dbm} related to t' if some other train t'' is matched to $\sigma(t)$. Finally, Constraints (8.15) make sure that the contribution is zero if departure $\sigma(t)$ is canceled.

8.5.2 Reducing the Model

In order to simplify (and streamline) the model description we treated every train as a linked arrival train. However, some arrival trains are not linked, and some trains are initial trains already in the system. For both types of trains the constraints can be simplified, and for an instance with few linked arrivals the model may be reduced substantially. The variables cat_t^i decide the category for a train t. For initial trains and non-linked arrivals the category is set as a problem parameter, and so for such trains Constraints (8.6) are not necessary. Constraints (8.7) can be removed and replaced by setting the upper bound to zero for any departure for which the known category of t is incompatible. Constraints (8.8)-(8.9) are unnecessary.

A non-linked arrival or an initial train t has no DBM or TBM contribution from any previous train. Constraints (8.14) can have the κ_t^{dbm} term removed, and we can set $c_{\sigma(t)} = 1$ (because instead train t will always take its own remaining DBM). Constraints (8.14)-(8.15) are unnecessary in such cases.

8.6 Column Generation Model

The formulation presented in section 8.5 models every initial and arrival train individually even if some of them constitute a sequence of linked arrivals and departures and therefore essentially represents a single physical train unit. In this section we present a formulation that models each physical train unit using a single variable. The number of possible variables grows more than exponentially by the number of present linked arrivals. Initially every train can be matched to any linked departure (i.e. departure with a linked arrival), and continue to any other linked departure, before it finally reaches its final departure. The number of potential paths is bounded by $O(|T| \cdot |L|! \cdot |D|)$, where $|L|$ is the number of linked arrivals/departures. We therefore propose a column generation approach for solving the model. Column generation is a well-described technique used with success for solving MIP problems, e.g. the Vehicle Routing Problem with Time Windows (VRPTW) [1, 6, 10]. It is assumed that the reader is familiar with column generation solution methods.

The model contains two types of variables. For every departure $d \in \mathcal{D}$ we introduce a binary variable c_d that indicates whether d is canceled or not. For every possible train unit pattern $p \in \mathcal{P}$

we have a binary variable λ_p that indicates whether pattern p is used or not. A pattern represents a sequence of linked arrivals and departures. The objective is formulated as a minimization of the number of unmatched departures and cost of missed train reuses :

$$\begin{aligned} \min & \sum_{d \in D} cancellationCost \cdot c_d \\ & - \sum_{u \in \mathcal{U}} \sum_{p \in \mathcal{P}} reuseCost \cdot \alpha_p^{dep_u} \cdot \beta_p^{arr_u} \cdot \lambda_p \\ & + reuseCost \cdot |\mathcal{U}| \end{aligned} \tag{8.16}$$

$$\sum_{p \in \mathcal{P}} \alpha_p^d \lambda_p \geq 1 - c_d \qquad d \in \mathcal{D} \tag{8.17}$$

$$\sum_{p \in \mathcal{P}} \alpha_p^d \lambda_p + \sum_{p \in \mathcal{P}} \phi_p^d \lambda_p \leq 1 \qquad d \in \mathcal{D} \tag{8.18}$$

$$\sum_{p \in \mathcal{P}} \beta_p^t \lambda_p + \sum_{p \in \mathcal{P}} \varphi_p^t \lambda_p \leq 1 \qquad t \in \mathcal{T} \tag{8.19}$$

$$\sum_{p \in \mathcal{P}} maint_p^{day} \cdot \lambda_p \leq maxMaint \qquad day \in \mathcal{H} \tag{8.20}$$

$$\lambda_p \in \{0, 1\} \qquad p \in \mathcal{P} \tag{8.21}$$

$$c_d \in \{0, 1\} \qquad d \in \mathcal{D} \tag{8.22}$$

The α_p^d is a binary coefficient that indicates whether departure d is covered by pattern p. The β_p^t is a binary coefficient that indicates whether train t is used by patten p. Therefore $\alpha_p^{dep_u}$ indicates whether departure $dep_u \in \mathcal{D}$ is covered by pattern p, and $\beta_p^{arr_u}$ whether train $dep_u \in \mathcal{T}$ is used by p. Finally, ϕ_p^d and φ_p^t are binary coefficients that indicate whether a departure d or train t is blocked as a result of pattern p. A train is blocked by a pattern if the final matching of the pattern ends (or terminates) on a departure that is linked to some arrival. No other pattern may use this arrival-train, as it would mean that two patterns are using the same physical train without keeping proper score on train type, DBM and TBM. A departure is likewise blocked by a pattern if it starts using a train of a linked arrival. This corresponds to using one of the replacement trains which assumes (or requires) that the linked departures was canceled. Essentially this means that the departure must be blocks if such a pattern is used. Finally, the coefficient $maint_p^{day} \in \mathbf{Z}^+$ indicates the number of maintenance operations performed on day *day* using pattern p.

Constraints (8.17) ensure that every departure is assigned (to some train), unless there is a cancellation. Constraints (8.18) ensure that at most one train is assigned to every departure, and also blocks departures for trains that assume that the departure is cancelled. Constraints (8.19) ensure that each train is assigned at most once. Trains are blocked by patterns if they use the corresponding linked departure. Constraints(8.20) ensure that the total number of maintenance operations (every day) is respected. For comparability it is here assumed that, given a matching (t,d), the day that a maintenance operation is performed is fixed. The day of maintenance operations can however be made a choice in the subproblem without difficulty. Finally, Constraints (8.21) and (8.22) show the variable domains. Note that Constraints (8.22) can be relaxed as Constraint (8.17) ensures that the variables are *naturally* binary in all feasible solutions.

The number of variables in the model is exponential by the number of linked departures, however compared to the MIP model present earlier the number of constraints is reduced to $O(|\mathcal{D}| + |\mathcal{T}| + |\mathcal{H}|)$.

For obtaining the optimal *integer* solution to this problem (using column generation) a Branch-and-Price (B&P) framework can be adopted. In our solution method we will however only add columns in the root node and use Branch-and-Bound (B&B) for finding the best solution using the generated columns. In general this produces solutions of high quality, but no guarantee of optimality can be given in general. The gap between the found solution and the LP solution (of

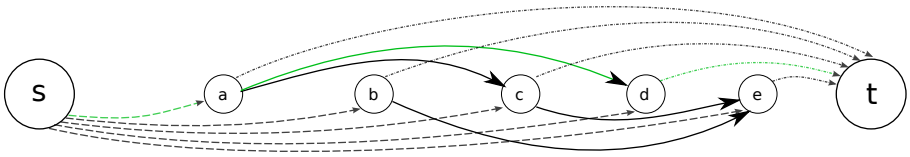


Figure 8.4: An illustration of the underlying graph for the matching subproblem. Every arc represents a match between a train and a departure. A feasible path (or linked sequence pattern) starts in the source s and ends in the sink/target t traversing a set of arcs. The feasible path (s, a, d, t) is shown in green. A linked sequence can start (or terminate) with many different trains (or departures) thus the dashed arcs illustrate where multiple arcs exist with the same origin and destination vertex.

the root node) gives a bound on the optimality gap. It is left as future research to develop a full B&P framework.

8.6.1 Column Generation Subproblems

We distinguish between generating two families of columns. The first family of variables consists of patterns containing only one train-to-departure matching. It is relatively easy to enumerate all choices in this family and produce columns with the most negative reduced costs. In our implementation we pre-generate all columns of this family. Therefore, for the next family of variables, assume that the following patterns consist of at least two train-to-departure matches.

The subproblem can be split into one subproblem per train category. This transformation will both simplify and reduce each individual problem as the number of compatible departures is lower. The complexity of a labeling algorithm is also reduced since it is no longer needed to keep track of the train category when extending arcs. An additional advantage is that all subproblems can then be solved in parallel.

The subproblem consists of solving a Resource Constrained Shortest Path Problem (RSCPP). The underlying graph consists of one node per linked arrival in addition to one source and one sink node, see Figure 8.4. The arcs constitute matching choices. Three types of arcs are added. First, arcs originating from the source to every node in the graph represent compatible trains that are matched to the linked departure of the node. Second, arcs are added between nodes that represent compatible linked continuations, i.e., linked arrival/departures that connect to another linked arrival/departure. An example: in the (s, a, d, t) path the departure of the initial train matching (represented by arc (s, a)) is linked to an arrival (node a). The departure of the next matching (arc (a, d)) is connected to another linked arrival (node d). The departure of the last matching is not linked to any arrival, and thus the sequence ends. As described earlier, some patterns (represented by the path) may block other trains or departures.

More formally, we construct a directed non-cyclic graph $G(V, A)$ for every category $c \in \mathcal{C}$. Let $v_\pi \in V$ denote the *source* vertex, and $v_\omega \in V$ denote the *sink* vertex. Finally, we construct one vertex $v_{la} \in V$ per compatible linked arrival $la \in \{a \in \mathcal{L} | c \in compCatDep_a\}$. The arcs in the graph represents a train and departure match. For every train t of category c ($\{t \in \mathcal{T} | cat_t = c\}$) an arc $a_{(\pi,la)} \in A$ connects v_π and to v_{la} if the distance between availability time of t and departure time of departure $\sigma(la)$ is sufficient[2]. Likewise for every departure d compatible with category c ($\{d \in \mathcal{D} | c \in compCatDep_d\}$) an arc $a_{(ld,\omega)} \in A$ connects vertex v_{la} with v_ω, again only if the necessary time is available. Finally for every pair of linked arrivals (la_1, la_2) an arc $a_{(la_1,la_2)}$ connects the train of la_1 to departure $\sigma(la_2)$ if it respects the required time. No edge directly connects the v_π to v_ω.

[2]This could depend on multiple factor such as expected/minimum routing time, maintenance time and dwell time. For the sake of simplicity and comparability reasons we only require that the train availability time is less (or equal) to the departure time.

Theorem 8.2. *Any path originating in π and terminating in ω represents a feasible matching.*

Proof. Every path consists of an initial arc and a terminating arc, and optionally multiple intermediate arcs. All arcs represent matchings that are possible to make. Since we only have one category per subproblem, the construction of the graph inherently ensures that the train category is compatible with the assigned departures. Since we assume that maintenance can be performed on any matching, the required DBM and TBM can be fulfilled. □

Theorem 8.3. *All possible matchings of at least two departures are valid paths in the constructed graph, originating in π and terminating in ω.*

Proof. All feasible matchings, w.r.t. category and time, are present in the graph. All initial matchings are represented as arcs originating in the source π. All intermediate matchings are present as arcs connecting the non-terminal nodes. Finally, all possible terminations of linked sequences are represented as arcs terminating in the sink ω. □

The subproblem can now be defined as the problem of finding the minimum cost path originating from v_π and to v_ω, given some edge costs, maintenance costs and restrictions. Every edge has a primal cost corresponding to the objective in the master problem, i.e., zero if no train reuses are satisfied or $-(n \cdot reuseCost)$ cost if n reuses have been satisfied by the path. A dual cost also appears on every edge that depends on the dual values given after solving the master problem in each iteration. The dual value of (8.17) and (8.18) are added to all arcs that include the corresponding departure. The dual of (8.19) is added to arcs including the corresponding train. The maintenance restrictions relate to the DBM and TBM restrictions. These values must be positive at all times, and all edges either increase or decrease these values. A path starts with values of 0 for DBM and TBM. All arcs originating from the source increase the values as indicated by the $remDBM$ and $remTBM$ on the train (of the arc). Other edges only decrease the values as indicated by $reqDBM$ and $reqTBM$ of the departures. Before extending an arc maintenance can be performed, which replenishes the DBM and/or TBM levels, but this comes at a cost of the corresponding dual of (8.20).

The problem is a RSCPP due to the maintenance constraints. In addition to the objective coefficients, the duals from Constraints (8.17) and (8.18) are added to arcs of the corresponding departure, and the duals from Constraint (8.19) are added to arcs of the corresponding trains. The appropriate dual from Constraints (8.20) is added every time a maintenance operation is scheduled.

The subproblem can be formulated as a mathematical problem:

$$min \sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{(i,j)\in A} \beta_{ij}y_{ij} + \sum_{(i,j)\in A} \alpha_{ij}z_{ij} \tag{8.23}$$

$$s.t. \sum_{(i,j)\in\delta^+(\pi)} x_{ij} = 1 \tag{8.24}$$

$$\sum_{(i,j)\in\delta^-(\omega)} x_{ij} = 1 \tag{8.25}$$

$$\sum_{(i,j)\in\delta^+(v)} x_{ij} = \sum_{(i,j)\in\delta^-(v)} x_{ij} \qquad \forall v \in V \setminus \{\pi, \omega\} \tag{8.26}$$

$$\sum_{(i,j)\in\Pi_v} reqDBM_{ij} \cdot x_{ij} + maxDBM \cdot y_{ij} \geq 0 \qquad \forall v \in V \tag{8.27}$$

$$\sum_{(i,j)\in\Pi_v} reqTBM_{ij} \cdot x_{ij} + maxTBM \cdot z_{ij} \geq 0 \qquad \forall v \in V \tag{8.28}$$

$$x_{ij} \in \{0,1\} \quad y_{ij} \in \{0,1\} \quad z_{ij} \in \{0,1\}$$

Where $\delta^+(v)$ and $\delta^-(v)$ respectively denote the outgoing and ingoing edges of vertex v. The binary variables x_{ij} define whether flow is used on edge (i,j) in shortest path, and y_{ij} and z_{ij} respectively determine whether DBM and/or TBM is performed on arc (i,j). Constraints (8.24)-(8.26) constitute the flow-conservation in a shortest path formulation. Constraints (8.27) and (8.28) ensure that the sufficient DBM and TBM is available - these levels can never be negative. Note that the graph is acyclic which means that we know all possible edges that can appear before reaching any vertex v - we denote Π_v as the set of all such edges. The constraints thus ensure that if a matching is made, then the DBM/TBM levels on any edge (leading up to v) must be non-negative.

Labeling Algorithm

As an alternative to solving the mathematical model of the subproblem directly, we propose a dynamic programming approach for finding the optimal paths for the resource constrained shortest path. We refer to Irnich [5] for a more in-depth description of a this topic.

The labeling algorithm is similar to a shortest path algorithm that uses full enumeration, e.g. using a Breath First Search (BFS) strategy, to find the minimum cost path. In addition, we also need to respect some side-constraints. In our method, a label is a partial path from the source to some intermediate vertex that also keeps track of the total reduced cost, remaining DBM, remaining TBM and performed maintenance. Every arch has a primal and dual cost and a required level of DBM and TBM.

Initially, we generate the empty label at the source. In every iteration of the labeling algorithm, we pick one label at some vertex v and extend it. When extending we generate new labels from every outgoing edges from v. Due to the possibility of performing maintenance, we generate multiple labels for every outgoing edge: one that does not perform any maintenance, one that only performs DBM maintenance, one that only perform TBM maintenance and one that performs both DBM and TBM maintenance. Labels are not extended if either DBM or TBM becomes negative when subtracting the required level of DBM and TBM since these represent prefixes of infeasible paths.

In order to reduce computational time, we introduce dominance rules. A label a is said to dominate another label b, if we can safely remove b without loosing optimality. By removing a label b we omit searching all paths that follow the matching pattern of b.

Theorem 8.4. *A label ($cost_a$, $remDBM_a$, $remTBM_a$) at vertex v dominates ($cost_b$, $remDBM_b$, $remTBM_b$) at vertex v if $cost_a \geq cost_b$, $remDBM_a \geq remDBM_b$ and $remTBM_a \geq remTBM_b$.*

Instance	Arrivals	Linked	Departures	Reuses
B1	1235	475	1235	804
B2	1235	475	1235	0
B3	1235	0	1235	0
B4	1780	722	1780	1089
B5	2153	720	2153	1089
B6	1780	722	1780	1089
B7	304	144	304	187
B8	304	144	304	187
B9	1967	860	1967	1226
B10	196	89	196	123
B11	1122	486	1122	726
B12	570	263	570	377

Table 8.1: A description of the tested instances. The columns show instance names, number of train arrivals, number of linked train arrival/departures, number of train departures and number of specified train reuses.

Proof. The first label has lower cost and more DBM and TBM remaining. The second label cannot have any advantages in terms of future matchings or maintenance costs. Thus, no matter how the path continues from v, the first label will always be at least as good as the second label. □

Finding the optimal path in the subproblem can be both time and space consuming. Since it is sufficient to find any one path with negative reduced cost, we will rely on generating heuristic columns initially. Only when no heuristic columns (with negative reduced cost) can be found, we solve using the exact labeling algorithm.

We adopt two variations of the full labeling algorithm to find heuristic negative reduced cost paths. The first variant allows no DBM nor TBM maintenance which efficiently limits the number of possible matchings. The second variant only allows at most one DBM and at most one TBM maintenance. This is motivated by the fact that one maintenance is likely sufficient when considering short planning periods, c.f., the provided data-set.

8.7 Benchmarks

The main characteristics of the tested instances are shown in Table 8.1. They correspond to the instances of the final phase of the ROADEF Challenge 2014. All tests are run on a dedicated machine with dual 2.66 GHz Intel Xeon E5345 processors and 24 GB of memory. Both processors have 4 physical cores supporting 2 threads per core. The IBM ILOG CPLEX version 12.5 optimization software is used for solving all Linear Programs (LPs) and MIPs. All solutions are verified to be correct (with respect to feasibility and solution cost) using a modified version of the provided *solution checker* [11]. In the modified version all constraints unrelated to the matching have been omitted.

The timelimit set in the ROADEF Challenge 2014 was 10 minutes. Within this limit the submitted algorithms had to perform both matching and routing within the station. The provided instances cover up to 7 consecutive days of arrivals and departures. The provided timelimit for the challenge seems a bit restricting given a planning horizon of several days. Due to the difficulty of the problem we target a time limit of 10-30 minutes.

An initial benchmark shows that both solution methods are unable to solve all instances to optimality within a few hours. Table 8.2 shows the details of the instances that were solved to

					Runtime in seconds			
Instance	Obj	Generated	Cons	Vars	Sub	LP	MIP	Total
B3	0	0	3 749	229 841	0.0	3.3	21.0	25.0
B7	4 400	15 629	946	19 048	33.4	28.2	20.3	83.0
B8	4 400	17 070	946	19 027	27.4	25.9	21.1	76.0
B10	2 100	3 413	620	7 050	0.7	2.4	0.2	3.0
B12	14 700	25 017	1 753	47 258	22 912.6	81.0	52.3	23 115.0

Table 8.2: Instances solved using column generation. The columns show instance name, objective of found solution, number of generated column, number of constraints and variables in final program, and finally total runtime of subproblem, LP solving, MIP solving and total. The omitted instances (B1, B2, B4, B5, B6, B9, B11) were not solved.

			Runtime(s)	
Instance	LP Relaxation	Columns	Subproblem	LP
B1	74 278	363 378	255.9	1 484.1
B2	32 596	633 486	214.9	1 532.0
B3	0	0	0.0	3.6
B4	134 582	619 406	264.2	1 485.9
B5	132 582	429 643	191.9	1 567.7
B6	133 095	652 439	270.3	1 477.5
B7	3 400	28 980	10.6	52.4
B8	3 400	30 242	12.0	55.7
B9	222 156	399 799	253.4	1 508.2
B10	1 800	6 168	1.1	3.4
B11	34 735	285 340	393.4	1 368.3
B12	13 503	71 922	1 533.2	173.6

Table 8.3: Root relaxations results given a time limit of 30 minutes. The columns show the instance name, objective of final relaxation, number of generated columns, and time spent generating columns and solving the LP relaxations. Optimal results were found for B3, B7, B8 and B10.

	Using Column Generation			No Column Generation	
Instance	Columns	Time	Objective	Time	Objective
B1	278 229	1 207	93 200	1 213	254 600
B2	383 937	1 200	61 000	1 201	228 000
B3	0	26	0	5	0
B4	300 143	1 162	209 300	333	379 000
B5	248 919	1 203	221 100	87	370 900
B6	285 498	1 201	211 800	341	379 000
B7	30 245	78	4 300	81	73 200
B8	27 942	86	4 400	81	73 200
B9	277 485	1 207	252 900	1 215	456 700
B10	5 908	5	2 000	26	42 400
B11	171 118	1 215	80 600	1 210	258 000
B12	71 442	1 206	14 300	1 205	135 600

Table 8.4: Solutions found using the column generation approach in 20 minutes. Results are compared to using no column generation, i.e., allowing no linked sequences.

optimality using the Column Generation Method (CGM). A few instances run out of memory before being able to solve the root relaxation to optimality. Given a high time-limit the MIP method is able to solve the same instanced except B12. We discuss details of the column generation later in this section.

Solving the root relaxation of the instances seems to be a hard in general. Given a time-limit of 30 minutes only four instances are solved, see Table 8.3.

8.7.1 Heuristic Results

Solving the problem instances using the exact methods proves to be very difficult and we therefore also investigate the performance of the methods in a heuristic context.

The next benchmark shows results obtained using CGM with a 20 minute time limit. We allocate 10 minutes for generating columns, followed by a 10 minute MIP solve (using CPLEX) of the master problem. Table 8.4 summarizes the results. The results are compared to no column generation, i.e., not allowing any linked sequences. It is observed that the result of disabling column generation seems to have a drastic effect on solution quality. This argues that ignoring linked sequences altogether is undesirable as it will drastically penalize the achievable objective. Solutions are now found for all instances in contrast to the exact approach, were only 4 instances where solved within the same timelimit. It is observed that the time is mostly spent solving LP relaxations, except for one case. Heuristic columns were used to speed up the subproblem process instead of solving the exact optimization problem in every iteration. Whenever no columns with negative reduced cost are found the method solves the exact problem. Preliminary results show that this approach is favorable as solving the exact subproblem is in many cases extremely time-consuming. Many columns are generated during the execution of CGM, and these are gradually increasing the master problem size. In future work, it might be interesting to remove unnecessary columns after a few iterations.

In our final benchmark we run the instances using the MIP Method (MIPM). As before we set a runtime limit of 20 minutes – the results are shown in Table 8.5. We note that this approach was unable to provide any solution for most instances. However, for instances B7, B8 and B10 MIPM was able to find slightly better results than CGM. A worse objective was found for the B12 instance. MIPM was unable to find solutions to all instances due to insufficient memory.

Instance	Objective	Constraints	Variables	Time
B3	0	1 151 899	706 138	1199.4
B7	3 400	119 708	42 400	31.3
B8	3 400	119 708	42 400	31.0
B10	1 000	72 160	20 440	56.0
B12	43 900	571 526	131 360	1204.5

Table 8.5: Solutions found using the MIP approach. This approach was unable to produce any feasible solution to the omitted (B1, B2, B4, B5, B6, B9, B11) instances.

8.8 Conclusions

We have described and investigated the Departure Matching Problem which is identified as a crucial subproblem of the RSUM problem in the ROADEF/EURO Challenge 2014. Without explicitly considering the matching problem, too many departures will be uncovered.

We prove in section 8.4 that the DMP is NP-hard in the strong sense by reduction from the 0-1 Multiple Knapsack Problem.

We have proposed two methods for solving the DMP. We first presented a pure MIP formulation of the problem which could act as a reference point for future studies. The model is however large in terms of variables and constraints and the benchmarks show that this model is unable to solve most of the proposed instances. Memory usage is one significant drawback of this method.

A second solution method based on column generation has also been presented. This model is simple and can without much difficulty be extended even further to handle more constraints. It is shown how the subproblem can be split into several independent problems thereby reducing complexity and enabling parallelism. The benchmarks for this approach show that we can find good solutions fast, if the method is used with time limits. However, even solving the root node relaxation is shown to be difficult in multiple cases. We expect that the method could be improved if embedded in a B&P framework with more efficient handing of the columns. Furthermore, the performance of this method could be further improved if a good initial solution can be provided as a *hotstart* to the CGM, e.g., the result of a heuristic method.

The considered data instances proved to be surprisingly hard to solve. The final results of the ROADEF challenge winners suggests that it is not possible to obtain a satisfactory solution to the overall problem, RSUM. There may not even exists a solution without cancellations.

For the sake of simplicity it has been assumed that all matches are possible where the departure time occurs after the arrival time. In reality it might be more realistic to remove matching options where the time between arrival and departure is either insufficient. Trains may arrival late, some time is required to perform routing inside the station and time is required to perform maintenance. Further, it may not be necessary to consider long matchings, e.g., an arrival on day 1 with a departure on day 7. Reducing the number of possible matchings will reduce the number of decision variables and constraints and likely improve the success-rate of solution methods. This may be a viable practical approach to ensure feasibility.

For future work there are a multiple matters worth considering. There are several column generation techniques that can be investigated to improve performance, e.g., dual stabilization, branch-and-price and reduced-cost fixing. The potential of heuristic methods for DMP is unknown. Such methods can potentially find good solutions fast, or even be used to speed up an exact approach. It would be interesting to consider new data instances where it is know that there exists at least one feasible solution without any cancellations.

Bibliography

[1] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[2] Richard Freling, Ramon M Lentink, Leo G Kroon, and Dennis Huisman. Shunting of passenger train units in a railway station. *Transportation Science*, 39(2):261–272, 2005.

[3] Jørgen Haahr and Simon Bull. A Math-Heuristic Framework for the ROADEF/EURO Challenge 2014. Technical report, The Technical University of Denmark, 2014.

[4] John E Hopcroft and Richard M Karp. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

[5] Stefan Irnich. Resource extension functions: properties, inversion, and generalization to segments. *OR SPECTRUM*, 30(1):113–148, 2008.

[6] Brian Kallehauge, Jesper Larsen, Oli B.G. Madsen, and Marius M. Solomon. Vehicle routing problem with time windows. In *Column Generation*, pages 67–98. Springer US, 2005.

[7] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.

[8] Leo G Kroon, Ramon M Lentink, and Alexander Schrijver. Shunting of passenger train units: an integrated approach. *Transportation Science*, 42(4):436–449, 2008.

[9] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[10] Marco E. Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[11] François Ramond and Marcos Nicolas. *Trains don't vanish! ROADEF EURO 2014 Challenge Problem Description*. SNCF - Innovation & Research Department, 2014.

Part III

Freight Yard Optimization

Chapter 9

A Matheuristic Approach to Integrate Humping and Pullout Sequencing Operations at Railroad Hump Yards

Jørgen Thorlund Haahr* Richard Martin Lusby*

*Department of Management Engineering, Technical University of Denmark,
Produktionstorvet, Building 426, DK-2800 Kgs. Lyngby, Denmark
jhaa@dtu.dk, rml@dtu.dk

[1] **Abstract** This paper presents a novel matheuristic for solving the Hump Yard Block-to-Track Assignment Problem. This is an important problem rising in the railway freight industry and involves scheduling the transitions of a set of rail cars from a set of inbound trains to a set of outbound trains over a certain planning horizon. It was also the topic of the 2014 challenge organised by the Railway Applications Section of the Institute for Operations Research and the Management Sciences for which the proposed matheuristic was awarded first prize. Our approach decomposes the problem into three highly dependent subproblems. Optimization-based strategies are adopted for two of these, while the third is solved using a greedy heuristic. We demonstrate the efficiency of the complete framework on the official datasets, where solutions within 4-14% of a known lower bound (to a relaxed problem) are found. We further show that improvements of around 8% can be achieved if outbound trains are allowed to be delayed by up to two hours in the hope of ensuring an earlier connection for some of the rail cars.

9.1 Introduction

This paper addresses the 2014 challenge posed by the Railway Applications Section (RAS) of the Institute for Operations Research and the Management Sciences. The challenge focuses on an important problem arising in the railway freight transportation industry known as the Hump Yard Block-to-Track Assignment (HYBA). In particular, the problem requires one to consider, and schedule, the movements of a number of rail cars through a *classification* or *hump* yard. The primary purpose of such a yard is to act as a consolidation point, where rail cars arriving over a certain time horizon on a number of *inbound* trains are rearranged, or classified, into groups of rail

[1]Accepted for publication in *Networks*, October 2015

cars sharing the same destination. These groups are then subsequently pulled out and combined to form new *outbound* trains, which remove rail cars from the classification yard. A classification yard typically consists of four main components: an arrival yard, a *hump*, a classification *bowl*, and a departure yard. During the processing of an inbound train each of its rail cars is pushed over the hump, and, under the influence of gravity, and with the use of switches, rolls to a pre-assigned classification track in the bowl. Bowls typically consist of multiple, parallel tracks of possibly different lengths, where partial outbound trains can be assembled before being pulled together to form outbound trains in the departure yard.

How to handle the steady flow of rail cars is of paramount importance to the efficiency of any classification yard. However, coordinating the processing of arriving inbound trains with the allocation of classification tracks and the assembly of outbound trains is not a trivial task. All processes within the yard are subject to a variety of different restrictions and, if scheduled poorly, can result in a situation where rail cars needlessly wait long periods of time before leaving on an outbound train. The aim of the HYBA is to process all cars in such a way that their average dwell time in the yard is minimized.

In this paper we present a novel solution approach for the HYBA problem. We propose a matheuristic based on a decomposition of the problem into three distinct, highly dependent subproblems. A matheuristic embeds mathematical programming techniques within a larger heuristic framework. For two of the subproblems exact optimization-based solution strategies are employed. These are coupled with a heuristic approach to the third, and the performance of the full heuristic is analysed using the official data sets. These data sets are of a practical size and are based on a typical North American classification yard; each considers a planning horizon of 42 days during which 702 inbound trains arrive (carrying in total 52,246 rail cars) to be processed. There are between 42 and 58 classification tracks on which to sort the rail cars, which comprise 46 different destinations. Furthermore, there are 18 daily outbound trains. An outbound train is scheduled to depart at the same time every day, and each destination is served by exactly one outbound train. On such data sets, the proposed approach obtains acceptable solutions, within 4-14% of a known lower bound (for a relaxed problem), for all data sets. We report run-times of at most 11 minutes.

To identify any bottlenecks at the considered classification yard we also perform a series of what-if analyses. For example, we discuss the effects of having longer classification tracks or more capacity on the outbound trains and compare the performance improvement to the base case. Finally, we show that substantial reductions of around 8% in average dwell time can be made if it is possible to allow outbound trains to be delayed by up to two hours. Given the relatively short run times of the approach, it is clearly evident that the proposed methodology is equally applicable at the strategic level planning decisions concerning railway classification yard design.

In what follows we describe aspects of the solution approach in more detail. We begin in Section 9.2 with a short summary of previous research in this area. Section 9.3 provides a formal description of the problem, while Section 9.4 introduces the models developed, together with their respective solution approaches. The performance of the approach is the subject of Section 9.5, where we present results on the benchmark instances provided. Finally, conclusions are drawn in Section 9.6.

9.2 Literature Review

The highly complex nature of hump yard planning makes it a perfect application for operations research methodologies. Not surprisingly, various studies have been conducted on problems similar in nature, but not identical, to the one considered in this paper. In this section we provide a brief review of the research that is available in the literature and, where relevant, point out any differences from the problem at hand. Furthermore, we restrict the review to those studies that

concern hump yard planning on a microscopic level only (i.e., planning the shunting movements within a single hump yard) since this is precisely the problem we address in this paper. For a general introduction to shunting within hump yards the reader is referred to [6, 10], while models that address hump yard planning at a macroscopic level (i.e., between different yards), commonly known as the *railroad blocking problem*, can be found in [1, 2, 18].

Methods for hump yard planning are typically categorized as being either *single* or *multi stage* methods, with the latter being by far the more researched. In a single stage method, each rail car arriving on an inbound train can be humped to a classification track exactly once before it is pulled out to an outbound train. Multi stage methods, on the other hand, allow rehumping (i.e., cars can be humped multiple times). This additional flexibility is provided in the hope of obtaining a better sorting of the cars and, ultimately, a more efficient use of the yard. Typically a so-called *mixing track* is specified, where any car assigned to the mixing track can be pulled back to the hump and classified again.

The works of [3, 4, 5] specifically address the problems of sorting and classifying rail cars at a hump yard using the mixed track concept. All papers restrict their attention to scheduling the classification bowl of the hump yard only. A noticeable difference between this problem and the one that we consider is that the formation of the outbound trains happens directly on the tracks of the classification bowl. Typically a track of the bowl is dedicated to a specific outbound train and the cars destined for the outbound train appear on the train in the order they are humped to the track. As reserving a classification track for a single outbound train claims significant sorting capacity, it is impractical to allocate an outbound train an entire track from the arrival of its first car until its departure. As such, a number of mixing tracks is used to temporarily hold cars for different trains. The cars on such tracks are then rehumped.

In [3], the authors describe an integer programming formulation that attempts to minimize the number of rehumpings that must be performed. These extra shunting movements are termed *roll-ins* by the authors. The model is solved using a branch-and-price algorithm and tested on 192 real-life instances from the Hallsberg marshalling (hump) yard in Sweden. The authors provide a direct comparison with a compact integer programming formulation and demonstrate the superiority of the column generation procedure. [4] extends this methodology to model situations in which the arriving rail cars each belong to a certain *block*, and these blocks must appear in a pre-specified order on the outbound train. Again, the Hallsberg hump yard in Sweden forms the basis of the computational study, where 50 instances with a planning horizon of three days are considered. An extension to [3] is also considered in [5], where a new arc-based model is presented, along with a rolling-horizon solution framework and an analysis of yard capacity.

The problem we consider shares strong similarities with that considered in [3, 4, 5]; however, there are several key differences. First, the incoming sequence of cars is not fixed in our problem (i.e., we can decide the order in which to process the inbound trains), we are not allowed to rehump cars, and we must coordinate the assembly of outbound trains using a set of pullback engines. That is, the assignment of a rail car to a classification track does not implicitly indicate the outbound train, nor the order in which it appears on an outbound train.

Identifying an efficient sorting of inbound rail cars at classification yards is also the focus of [8]. The authors adopt a more theoretical approach to the problem and prove that the problem of finding the minimum number of tracks required to sort a set of arriving rail cars into blocks of cars that can be pulled out in a specific order from the classification tracks is NP-complete. This topic is also addressed by [7, 12, 17]. In [12], the authors develop a novel encoding strategy for classification schedules, discuss its complexity, and present algorithms that can be used to solve practical rail car classification problems. Theoretical aspects of rail car classification are also considered in [17]. In addition, the author describes several practical extensions of the problem, and an integer programming formulation is developed to solve the classification problem. Finally, [7] completes an open proof from [12] and show that identifying optimal classification schedules in constructing one long outbound train from multiple inbound trains is NP-Hard.

Dirnberger and Barkan [9] consider improving the performance of classification yards and introduce the concept of so-called *lean railroading*. This approach adapts production management strategies to the railroad environment. The *pull-down*, or outbound train assembly, process is identified as the main bottleneck, and the authors suggest that to improve the performance of classification yards emphasis should be placed on identifying quality sorting strategies instead of merely measuring the number of cars processed at the hump. Studies reported in the paper suggest that capacity for train assembly can be increased by as much as 26%.

Attempts to improve the connection reliability of hump yards are provided in the two part series of papers [13, 14]. Determining which cars to process at the hump, and in which order, is critical in ensuring that cars meet specific (i.e., the earliest) outbound departures. The first paper [13] considers the relationship between priority-based-classification and dynamic car scheduling to produce a reliable service. The author emphasizes the need for better information to be available at the time at which a car is humped (ideally the outbound train to which it will be assigned). This can be coupled with a more efficient block-to-track assignment to ensure that the classification yard is being used to its full potential. This is precisely the topic of the second paper [14]. The author describes a dynamic car block-to-track assignment strategy based on delivery commitment rather than a first-in-first-out strategy. In other words, cars with very little schedule slack should have access to the first available outbound train capacity. The proposed heuristic framework sorts cars by outbound train and destination yard block as opposed to just destination yard block, giving greater knowledge regarding the exact make-up of each outbound train.

He et al. [11] present a Mixed Integer Program (MIP) model for optimizing single stage hump yard operations (i.e., no rehumping), from inbound train classification to outbound train assembly and departure. The model also appears to account for outbound block standing orders and limits on the number of pullback engines available to do the sorting. Due to its size and complexity, the authors present a decomposition-based heuristic and discuss its performance on several practical instances arising in China. The problems considered up to 170 inbound trains per day (with up to 8,000 cars per day in total). The objective essentially minimizes a combination of the dwell time of the rail cars in the yard and delays to outbound trains. Running times of the algorithm are reported to be within 10 minutes.

Finally, simulation models for hump yard planning are described in [15, 16]. The focus in these papers is not on optimizing the hump yard schedules, but rather in identifying bottlenecks in the existing infrastructure (i.e., the number of cars that can be handled) with existing scheduling strategies.

9.3 Problem Description

The considered problem has been introduced and defined by the Railway Application Section Competition 2014 [19]. In this section we present a standalone definition of the problem as well as introduce the notation we use throughout the paper. We reuse the notation and concepts of the original formulation to a great extent. An illustration of the problem mechanics is shown in Figure 9.1.

We assume that there is a set of inbound trains $\mathcal{I}$ arriving at the hump yard over a given time horizon. Each inbound train consists of a coupled, ordered sequence of *railcars* that will be separated at the hump in the yard. Each train $i \in \mathcal{I}$ arrives at the hump yard at a given time a_i, given in seconds from the start of the horizon. Note that this does not indicate the time at which it will be humped; a train can wait in the arrival yard as long as is necessary. The arrival yard is assumed to have infinite capacity. We denote the set of railcars by $\mathcal{C}_i$ for a given inbound train $i \in \mathcal{I}$. Naturally, the full set of cars to be processed is therefore $\mathcal{C} := \bigcup_i \mathcal{C}_i$. Along with a length l_c, each car $c \in \mathcal{C}$ has a known *block ID* (henceforth referred to as just block), denoted b_c. This indicates the car's next destination: for example *BOS* for Boston and *PHX* for Phoenix.

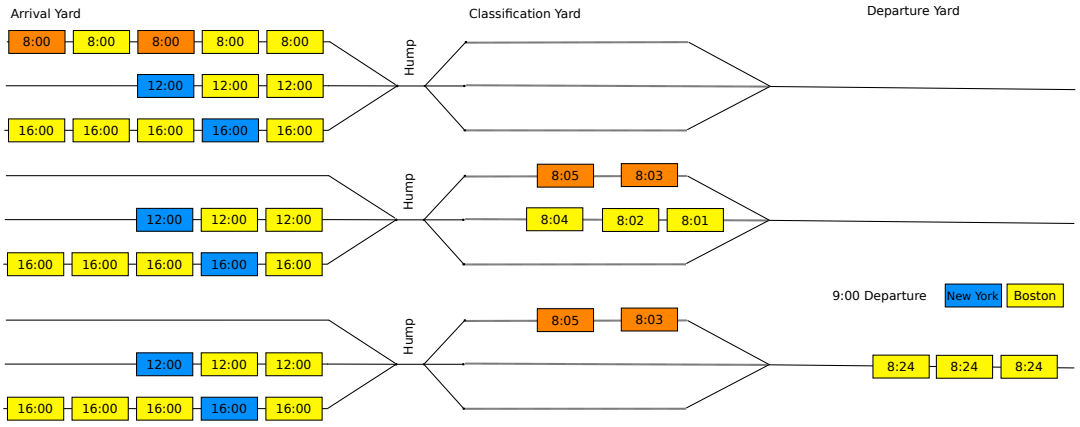


Figure 9.1: An example of a problem instance illustrating the railcars, the three different yards in three timesteps on the vertical axis. Railcars are represented as boxes with colours corresponding to their assigned destinations (or block ids). Assume one car can be processed by the hump every minute, and that the pullout time is 20 minutes. Three different inbound trains with distinct arrival times (8:00, 12:00 and 16:00) and one outbound train departing at 9:00 are shown. The top portion shows the initial state, where nothing is assigned. The middle portion shows an inbound train has been processed and bowl tracks have been assigned to the railcars. The bottom potion shows a cut has been pulled out and assigned to meet the departing train.

The set of all blocks is given by $\mathcal{B}$. As an example, Figure 9.2 indicates the composition of the first 10 inbound trains. In other words, the amount of cars of different outbound blocks on each train. We use different colors to distinguish between blocks. The bar chart provides a summary of the block counts only. On any given inbound train, the cars associated with a certain block are not usually in consecutive order, but distributed throughout the train. This random pattern is representative for the inbound trains in our dataset, i.e., we rarely see long sequences of cars with identical blocks.

Every inbound train $i \in \mathcal{I}$ must be processed, i.e., each of the cars $c \in \mathcal{C}_i$ must be decoupled, pushed over a hump, and moved to an available track in the classification bowl. Note that the process can be paused for a period, i.e., a partially decoupled inbound train can be stopped on the hump, e.g., if no track with sufficient space is available in the classification bowl.

It is assumed that the bowl consists of a set of parallel tracks $\mathcal{T}$, where each $t \in \mathcal{T}$ has a known length γ_t, and operates as a first-in-first-out queue. The tracks of the bowl are be used to sort the cars into *lines* of cars. A line is a sequence of cars of the same block and which are assigned to the same pullout (i.e., they appear on the same outbound train).

When humping a train several constraints must be respected. First, it is assumed two hump engines are available to perform the humping operations; one pulls cars from an inbound train to the hump, while the other is used to retrieve inbound trains from the arrival yard, i.e., bring them to the humping point. An inbound train's *humping time* refers to the time at which its first car is humped and two consecutive humpings must therefore be separated by some minimum duration. This duration is equal to either the minimum time required to hump all of the first train's cars (each car is assumed to take a constant time, λ seconds, to hump), or the time required to retrieve the next inbound train from the arrival yard, again constant and equal to δ seconds, whichever is the longer. In addition, all of a train's cars must be humped before any other train can be humped, i.e, no train can be partially humped; it is considered to be an atomic operation.

Railcars, that are coupled together, are pulled from bowl tracks and appended to an awaiting outbound train. Each outbound train $o \in \mathcal{O}$ is scheduled to depart at the given time d_o and has

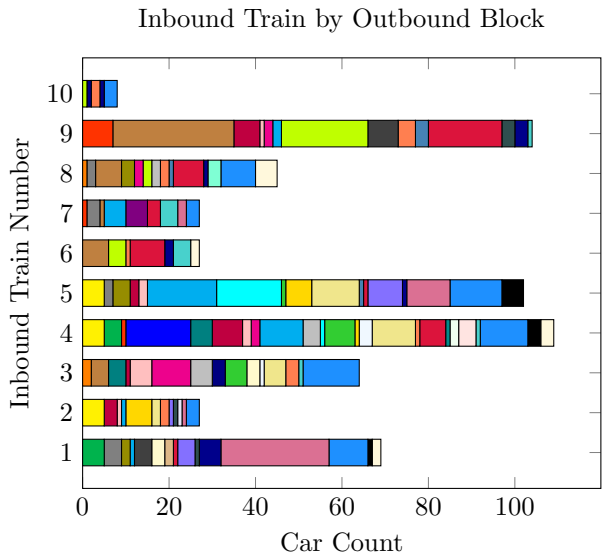


Figure 9.2: The different outbound blocks on each of the first 10 inbound trains. Each outbound block is associated with a specific colour.

a maximum length η_o of railcars that it can carry. An outbound train has a predetermined route through specific destinations (i.e., blocks) which is why the assigned railcars must conform to a *standing order*, that stipulates the order in which the railcar destinations (blocks) must appear on the outbound train.

Outbound trains are built in the departure yard using three available *pullout* engines. Pullout engines move so-called *cuts* of rail cars from the bowl to the departure yard. A cut simply refers to a sequence of lines that adhere to the standing order of the designated outbound train. Like hump engines, several restrictions exist for the pullout engines. First, pullouts from the same bowl track must be separated in time by a minimum duration to allow a smooth operation. Second, pullouts to the same outbound train must also be separated in time as multiple engines cannot build the same departure train simultaneously. Third, consecutive pullouts by the same engine must also be spaced by a minimum duration, corresponding to the time required to perform one job, which we denote ρ.

Finally, an outbound train can only be built within a certain time window of its scheduled departure time; e.g for the considered problem, an outbound train can start to be assembled, at the earliest, 4 hours prior to its departure. However, we also consider an extension to the problem where departures can be delayed. Note that delaying a given outbound train will increase the dwell time for all cars assigned to that departure; however, it will also potentially reduce the dwell time of other rail cars that would otherwise miss the connection if it departed on time.

The objective of this problem is to determine the humping sequence of the inbound trains, which bowl track to use when humping each car, and the schedule for the pullout engines (i.e., how each outbound train should be built) such that all constraints are satisfied and the average dwell time of the cars in the yard is minimized. In other words, we need to determine an itinerary for each individual railcar (i.e., hump time, assigned bowl track, pullout time and outbound train), such that none of the constraints are violated.

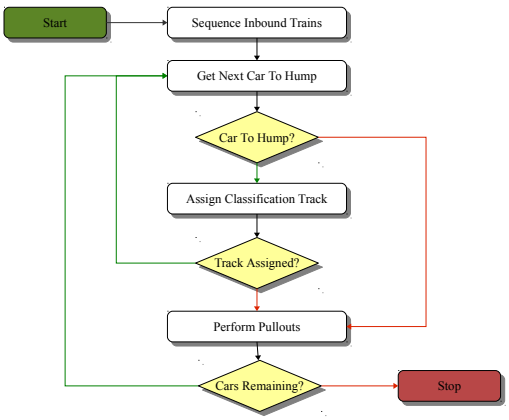


Figure 9.3: An overview of the proposed solution framework.

9.4 Modelling & Methodology

Given the problem's size and complexity, it is extremely difficult to construct a tractable mathematical model for the entire problem. Hence, we decompose the problem into three smaller, interdependent subproblems. which we term the Hump Sequencing Problem (HSP), the Block to Track Assignment Problem (BTAP), and The Pullout Allocation Problem (PAP), respectively. For the HSP and the PAP we describe MIP based optimization approaches, while we present a simple greedy heuristic for the BTAP. In this section we elaborate on each of these problems as well as the proposed methodolgy to solve them.

To set the context we provide a brief overview of the proposed methodology before going into specific details regarding each of the subproblems. Figure 9.3 illustrates the flow of the proposed approach. We begin by finding an arrival sequence for the inbound trains, e.g., using the HSP, and this remains fixed for the remainder of the algorithm. That is, after finding this processing sequence we never revise the humping order of the inbound trains. Cars are then iteratively humped into the bowl and assigned classification tracks using the BTAP. As soon as a car cannot be humped into the classification bowl, possibly due to a lack of space or no free tracks, the humping process is halted and pullouts are scheduled to make space in the bowl. Which pullouts to perform are decided by the PAP. Note that during an iteration the PAP and BTAP are not necessarily considering the bowl at the exact same time or time period. In general when the BTAP pauses at time t, e.g., due to lack of space or compatible tracks, the PAP will consider pullouts that occur before t.

This process of humping cars and performing pullouts continues until there are no cars left to process. To provide a quality measure on the solutions found, in Section 9.4.1, we show how one can obtain lower bounds on the minimum average car dwell time. The lower bounds, albeit potentially weak, provide some sort of quality measure for the obtained solutions. Sections 9.4.2-9.4.4 are dedicated to the HSP, BTAP, and the PAP, respectively.

9.4.1 Lower Bounds

In order to verify the quality of the solutions produced by the matheuristic it is important to obtain a lower bound on the total dwell time. Here we describe two rather simple approaches for generating such bounds. Both bounds assume that outbound train departure times are fixed.

The first lower bound assumes that all outbound trains have infinite capacity, that all trains can be humped immediately, and that there are neither capacity nor ordering restrictions in the

classification bowl. It does, however, respect the humping rate of the cars; the time at which any car is assumed to be available for a departure is a certain duration after the arrival time of the train it is on. This duration is the time needed to hump the cars ahead of it on the train.

The lower bound on average dwell time (in hours) is calculated as follows:

$$LB = \frac{(\sum_{i \in \mathcal{I}} \sum_{c \in \mathcal{C}_i} d_c - a_i)}{3600 \cdot |C|}, \tag{9.1}$$

where the d_c refers to the departure time of $c \in \mathcal{C}_i$. Here d_c is simply the departure time of the earliest outbound train car c can be assigned to such that $d_c \geq a_i + \lambda \cdot (n-1)$, where n states the position of car c in the sequence of cars on train i and ranges from one to $|\mathcal{C}_i|$.

The second lower bound is calculated similarly, with the exception that outbound train capacity is taken into consideration. That is, d_c denotes the earliest outbound train with available capacity car c can be assigned to. The method processes cars ordered by their lengths and (for the ease of computation) allows fractional cars to be assigned to outbound trains. Naturally, the second bound is likely to be tighter than the first; however, the magnitude of this increase can provide insight into how restrictive the outbound train capacity is.

The bounds are calculated independent of the details of the classification bowl, and it will therefore not vary across the data sets we consider. Intuitively one would expect a more accurate bound to be higher in cases with fewer classification tracks since processing the cars in the bowl would likely require more time. Nevertheless, calculating the two bounds using Equation (9.1) yields values of 12.046 and 12.575 dwell hours in average, respectively.

Formulating and solving the second bound approach as a MIP, thereby removing fractionality, improves the bound to 12.591 hours at the cost of a 10^4 factor increase in runtime.

9.4.2 The Hump Sequencing Problem

The hump sequencing problem entails identifying the *best* order in which to process the inbound trains. Depending on its block composition, it may or may not be critical to hump the cars on an inbound train into the bowl immediately. If, for example, the train is carrying cars for which the earliest outbound train is some time away, it may be preferable to hump another inbound train which arrives later but which carries cars for an earlier outbound train. Hence, simply processing inbound trains in their arrival order may result in some cars missing their earliest departure, and thus incurring unnecessary dwell time. We formulate this problem as a MIP and attempt to minimize the departure day of the final car to be processed. First, binary variables x_{co} are introduced and indicate whether or not car $c \in \mathcal{C}$ is assigned to outbound train $o \in \mathcal{O}_c$. $\mathcal{O}_c$ indicates the set of outbound trains to which car c can be assigned, i.e., $\mathcal{O}_c = \{o : o \in \mathcal{O}, b_c \in \mathcal{B}_o\}$. A second set of binary variables y_{ij} governs the sequencing order of inbound trains i and $j \in \mathcal{I}$. In other words, the variable y_{ij} takes the value one if train $i \in \mathcal{I}$ is humped before train $j \in \mathcal{I}$, and is zero otherwise. Note that for any two trains only one binary sequencing variable is required, and we use the arrival time of the trains to define a partial order. More specifically, we say $i \prec j$ if train $i \in \mathcal{I}$ arrives before train $j \in \mathcal{I}$. Finally, continuous variables h_i are defined and represent the humping time of train i. A solution to this problem must respect several constraints. In particular, a minimum separation time must elapse between consecutive humpings, no car can depart before it has been humped, and the length of departing outbound trains must be respected. This problem does not consider the classification bowl nor the pullout engines explicitly and thus is expected to provide an optimistic solution; however, for problems with many classification tracks (i.e., more than $|\mathcal{B}|$) it should provide a good indication of the processing order. The full mathematical model is given below.

$$\text{minimize: } \sum_{c\in\mathcal{C}}\sum_{o\in\mathcal{O}_c} day(o)\cdot x_{co}, \tag{9.2}$$

$$h_j \geq h_i + \Delta_{ij} - M(1-y_{ij}) \qquad \forall i,j\in\mathcal{I}, i\prec j, \tag{9.3}$$

$$h_i \geq h_j + \Delta_{ji} - My_{ij} \qquad \forall i,j\in\mathcal{I}, i\prec j, \tag{9.4}$$

$$\sum_{o\in\mathcal{O}_c} d_o x_{co} \geq h_{i(c)} + ind(c)\cdot\lambda \qquad \forall c\in\mathcal{C}, \tag{9.5}$$

$$\sum_{c\in\mathcal{C}_o} l_c x_{co} \leq \eta_o \qquad \forall o\in\mathcal{O}, \tag{9.6}$$

$$\sum_{o\in\mathcal{O}_c} x_{co} = 1 \qquad \forall c\in\mathcal{C}, \tag{9.7}$$

$$x_{co}\in\{0,1\}, \quad \forall c\in\mathcal{C},\ o\in\mathcal{O}_c, \tag{9.8}$$

$$y_{ij}\in\{0,1\}, \quad \forall i,j\in\mathcal{I}, i\prec j, \tag{9.9}$$

$$h_i, \geq 0, \qquad \forall i\in\mathcal{I} \tag{9.10}$$

where $day(o)$ gives the departure day of outbound train o, $i(c)$ gives the inbound train carrying car c, $ind(c)$ gives the index of car c in $\mathcal{C}_i$, and $\mathcal{C}_o$ denotes the set of cars that can be assigned to outbound train $o\in\mathcal{O}$. The objective (9.2) minimizes the total number of days require to process all cars. Constraints (9.3) and (9.4) collectively ensure a minimum time separation between two consecutive humpings. Here $\Delta_{ij} = \max(\lambda\cdot|\mathcal{C}_i|,\delta)$ which ensures a minimum separation time and enough time to hump cars of inbound train i, where M is a sufficiently large number. In this case M is equal to the latest arrival time of any train $i\in\mathcal{I}$. Constraints (9.5) stipulate that a certain time must elapse upon arrival before the car can depart on an outbound train. The maximum length of all outbound trains is enforced by constraints (9.6). This ensures that we cannot assign more rail cars to an outbound train than the outbound train can accommodate. Constraints (9.7) ensure that each car is assigned to exactly one departure. Finally, variable domains are given by constraints (9.8)–(9.10).

Due to the size of this model, it is solved using a kind of rolling time horizon based approach where each horizon has its own HSP and only considers a subset of arrivals. A rolling time horizon is justified in this context as any deviations from the arrival order when humping the inbound trains are likely to be local. For instance, it is quite unlikely trains will wait many days to be processed. By considering only a subset of arrivals at a time, we essentially limit the number of train ordering decisions that must be considered (thus reducing the number of y_{ij} variables). This is because only the order of the trains in the subset under consideration can be modified. We therefore solve Model (9.2)–(9.10) as a sequence of smaller HSPs of the same form, where $\mathcal{I}$ is limited to m trains. The subsets are generated based on the arrival order of the inbound trains. For example, the first HSP is solved using the first m trains to arrive. To ensure a cohesive transition between the sequence of smaller HSPs, the sets of inbound trains used to define two successive HSPs intersect slightly. For example, if m is 100, then after solving the first HSP, the solution found for the first 80 trains might be fixed. The second HSP then considers sequencing train 81 to train 180, and so on. Naturally, the capacity of the outbound trains is revised as we successively solve the HSPs due to any previous rail car assignments that have been fixed. In Section 9.5 this strategy is compared against the greedy approach of simply processing trains in their arrival order.

Finally, it is important to note that the solution to this problem might not be implementable in practice. Since the model does not consider classification tracks nor the pullout operations, there is no guarantee rail cars can actually leave on the departure to which they have been assigned in the HSP. The model is only used to provide an indication of hump sequencing order and is potentially quite optimistic.

9.4.3 The Block-To-Track-Assignment Problem

Whenever a rail car is humped, it must be assigned one of the classification tracks in the bowl. To assign a classification track we use a simple greedy heuristic. This procedure is as follows. If a line of the same block as that of the car being humped already exists in the bowl, and there is enough remaining capacity on the track, the car is humped to that track. Otherwise, a randomly selected "open track", i.e., an empty bowl track, is assigned. If neither option is possible, humping is momentarily paused, and pullouts are performed to create more space in the bowl.

Given the highly fragmented nature of the outbound blocks arriving on inbound trains, creating mixed tracks (i.e., classification tracks with possibly several different outbound blocks) would result in many short lines on the classification tracks. This in turn would lead to a greater number of pullouts being required to assemble the outbound trains. Thus, the greedy track allocation strategy ensures classification tracks are dedicated to specific blocks, possibly at the expense of more pauses in the humping of the rail cars. The performance of this simple heuristic strategy was compared to a more intelligent approach of assigning tracks based on outbound block volumes (classified as low, medium, or high based on number of cars). However, the latter, surprisingly, did not perform better.

9.4.4 The Pullout Allocation Problem

After humping has been performed in the framework, pullouts must be performed. Two pullout methods are developed and discussed. The first is a greedy approach (*Greedy*), while the second uses more sophisticated modelling techniques to better exploit the available pullout engine resources.

Greedy Pullout

The input for *Greedy* is a point in time, the bowl state at this time, and the next outbound train to process. *Greedy* sequentially processes outbound trains by departure time. For a given train $o \in \mathcal{O}$, it analyses all bowl tracks and tries to pull lines from tracks in the order specified by the departure's standing order. An overview of the method is shown in Algorithm 3.

The track with the longest available length is identified and pulled out (*LongestTrack*). The exact pullout time is determined by inspecting existing engine usage and pullouts. The *LongestTrack* sub-procedure ensures that *Greedy* finds the earliest, feasible time for a pullout. After each pullout the *time* is advanced by the pullout time, since this is the minimum time required to do the next pullout. Although it is possible, the algorithm will never pull the same track more than once for the same departure. Note that we use an inner while loop in order to be able to pull out several tracks with the same block.

Pullout MIP

One shortcoming of the *Greedy* method is its inability to share pullout engine resources and to consider what is beneficial or harmful for other outbound trains since it does not consider future consequences of local decisions. Here we propose a MIP-based pullout scheme that considers

Algorithm 3

```
1: procedure GreedyPullout(departure, bowl, time)
2:     for b ∈ BlockStandingOrder(departure) do
3:         (track, len) ←LongestTrack(bowl, b, time)
4:         while len > 0 do
5:             next ← NextPulloutTime(track,
                       departure, time)
6:             bowl ← PerformPullout(bowl, track,
                       departure, next)
7:             time ← next
8:             (track, length) ←LongestTrack(bowl,
                           b, time)
```

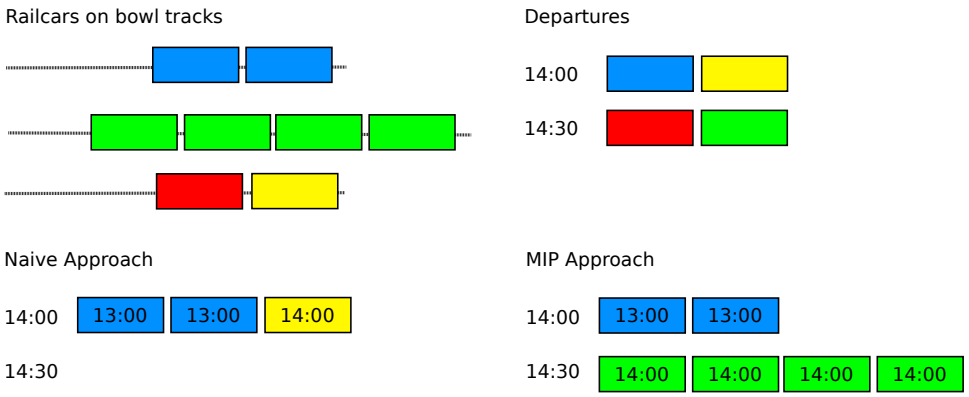


Figure 9.4: A comparison of the *Greedy* and MIP pullout approach. In the top left portion the current bowl tracks are illustrated, and in the top right portion the scheduled departures are depicted. In this example one pullout engine is available and requires one hour per pullout job, and each departure can be built two hours in advance. The *Greedy* method processes the departures iteratively by departure time, thus in this example nothing is assigned to the second departures as the engine has been fully assigned up until 14:00. The MIP method, however, identifies a solution that shares the engines such that the total number of pulled railcars is maximized.

multiple outbound trains simultaneously. Figure 9.4 compares the methods using a simple example.

At any point in time each bowl track contains a set of lines. We will consider the line closest to the pullout end of any track; any line behind this will be left untouched. Therefore, at any time, each track $t \in \mathcal{T}$ can be mapped to a unique block and, consequently, a unique departure (and unique standing order therein). The mapping is not bijective since a standing order can be mapped to multiple compatible tracks. The input for this method is hence a point in time, *time*, and the corresponding bowl state. Based on this input and the instance data, we formulate a MIP model to identify which pullouts to perform. This MIP forms the basis of Algorithm 4.

For each track $t \in \mathcal{T}$ a list of possible pullout candidates C_t is generated. All candidates for the same track differ in pullout time, effectively limiting how many cars can be pulled. The set of all candidates is denoted $C := \bigcup_{t \in \mathcal{T}} C_t$.

A binary decision variable x_c is introduced and indicates whether or not candidate $c \in C$ is selected. Some candidates cannot be selected simultaneously as this would create a standing order violation on the designated outbound train. We define set $C^- \subseteq C \times C$ to contain exactly these pairwise conflicts. C^- is determined in a preprocessing step.

If a candidate is selected we do not enforce all cars in the candidate to be pulled, only a subset. Therefore, for each track $t \in \mathcal{T}$ we monitor the total length of the cars pulled. To do this we introduce one continuous variable y_t for each track $t \in \mathcal{T}$. Note that the decision variables x_c and y_t can lead to an infeasible decision as it may imply taking a percentage of some car. In practice we therefore leave the one fractional car behind. In addition, since all cars have individual lengths, these decision variables do not factor in the railcar by length ration. Optimality is thus no longer guaranteed, but we argue that a significant speed-up is achieved, through a simplified and smaller model, at a low cost. A fractional solution is only achieved when the outbound train has reached its capacity; this is rarely the case. Also note that obtaining an optimal solution here using a more precise model need not make the overall framework optimal.

We also define the set $\mathcal{T}_o \subseteq \mathcal{T}$, which gives the set of tracks containing lines for outbound train $o \in \mathcal{O}$. To accurately model the pullout movements two types of (time) interval sets are introduced. The first, Π, is the set of non-overlapping intervals generated by including all candidate pullout start and end times. An element, π, of this set is a subset of C that overlaps with the interval. A similar second set, Ω_o, contains all non-overlapping intervals generated by including all candidate pullout start and end times corresponding to a outbound train $o \in \mathcal{O}$. An element, ω, of this set is a subset of C that overlaps with the interval.

As mentioned in Section 9.3, outbound trains have a given capacity. Given the current bowl state and previous pullouts the remaining available length $remCap_o$ can be computed for all outbound trains. Finally, it is also given that we have a number, $numEngines$, of pullout engines available. The number of pullout engines being used, as a result of earlier choices, during interval $\pi \in \Pi$ is assumed to be $usedEngines_\pi$.

The pullout allocation MIP is as follows:

$$\text{maximize:} \sum_{t \in T} y_t \cdot obj_y \tag{9.11}$$

$$\sum_{c \in C_t} x_c \leq 1 \qquad \forall t \in T \tag{9.12}$$

$$x_c + x_{c'} \leq 1 \qquad \forall (c, c') \in C^{-} \tag{9.13}$$

$$\sum_{t \in \mathcal{T}_o} y_t \leq remCap_o \qquad \forall o \in \mathcal{O} \tag{9.14}$$

$$y_{track(c)} \leq x_c \cdot length_c + M(1 - x_c) \qquad \forall c \in C \tag{9.15}$$

$$y_t \leq \sum_{c \in C_t} x_c \cdot length_c \qquad \forall t \in T \tag{9.16}$$

$$\sum_{c \in \omega} x_c \leq 1 \qquad \forall \omega \in \Omega_o, o \in \mathcal{O} \tag{9.17}$$

$$\sum_{c \in \pi} x_c \leq numEngines - usedEngines_\pi \qquad \forall \pi \in \Pi \tag{9.18}$$

$$x_c \in \{0, 1\} \qquad \forall c \in C \tag{9.19}$$

$$y_t \geq 0 \qquad \forall t \in T \tag{9.20}$$

The objective is ideally to maximize the number of cars but as discussed we only model the number of feet pulled out. In this model we approximate the number of cars by setting obj_y to the average number of cars per feet. This approximation works well due to the fact that the capacity of most outbound trains is not binding. And, in any case, all cars must be pulled out sooner or later.

The first set of constraints (9.12) ensure at most one candidate is selected per track. Constraints (9.13) guarantee that pullouts respect the outbound train's standing order by prohibiting conflict pairs. The remaining capacity of an outbound train must be respected; this is the purpose of Constraints (9.14). Constraints (9.15) link the pullable length of the track to the selected candidate decision, while (9.16) makes sure that zero length is pulled from a track if no candidates from the track are selected. Constraints (9.17) ensure candidates selected for the same departure, o, respect a minimum distance. This is modeled by making sure that one candidate can remain active (in a pullback process) in the same interval for the same departure. Similarly constraints (9.18) ensure that at most $numEngines$ engines are used concurrently in every period. Finally, variable domains are given by (9.19) and (9.20).

Framework Integration

This MIP-based procedure is called every x minutes at time $t = t_{last} + x$, where x is set to an appropriate value set in the tuning phase. The pullout MIP identifies the best set of pullouts to be performed; however, not all are actually performed by the overall framework since the model only considers partial information based on the current state of the bowl. Ideally, one only wants

to perform pullout decisions that will definitely not change in later iterations,. In our case, this corresponds to every decision made before the last hump time or before time t. An overview of the algorithm is given in Algorithm 4. Using a discretization step all candidate pullouts after *time* within the build window of the departures are generated. Next the list of candidates is filtered. All candidates that violate standing orders due to existing pullouts in the bowl are removed. Candidates that violate the separation constraint (of past assignments) on the corresponding track are removed. Candidates that violate the separation constraint (of past assignments) on the departure are also removed. The generated candidate pullout events and existing engine usage are analyzed and the available engine capacity is stored for all relevant time periods.

Algorithm 4

```
1: procedure PulloutMIP(bowl, time, time')
2:     C ← {}
3:     for t ∈ ClassificationTracks do
4:         s ← FrontSegment(t, bowl, time)
5:         b ← BlockOfSegment(s)
6:         d ← DepartureOfBlock(b)
7:         (w_α,w_ω) ← DepartureBuildWindow(d)
8:         C ← C∪ Filter(Generate(t,w_α,w_ω))
9:     model ← BuildMipModel(bowl, C)
10:    pullouts ← Solve(model)
11:    lastHump ← LastHumpedCar(bowl)
12:    for p ∈ pullouts do
13:        pt ← PulloutTime(p)
14:        if pt ≤ max(lastHump, time') then
15:            bowl ← PerformPullout(bowl, p)
```

An example illustrating the pullout process in the solution framework is shown in Figure 9.5. For the sake of argument we make a few simplifications. First, assume that every outbound train only has a standing order of one block type. Second, assume that only one pullout engine exists and requires one hour to perform one pullout. Third, assume that a outbound train can be built two hours in advance. Finally, assume that all rail cars are of equal length. The example shows the relation between the decisions made by the MIP (Algorithm 4) and the decisions actually implemented by the solution framework. Only a subset of the the MIP decision are adopted by the framework as it only considers partial information. Note, in Figure 9.5 at 12:00 the MIP recommends pulling two blue rail cars at 12:00-13:00 but in the next iteration this decision has been changed (to pull three red rail cars instead) as more rail cars have arrived in the bowl. Also note, in the second iteration, that the decision to pull three red rail cars at 13:00-14:00 is changed to pull four rail cars from the same track in the third iteration.

This approach is superior to the *Greedy* approach as it considers interdependencies between multiple departures. Although the method operates on partial future information, depending on how much is in the bowl, it can still consider future consequences of pullouts to some extent. Solving the MIP model using a commercial solver allows us to find near optimal solutions very quickly in practice. This benefit is, however, also a liability since the runtime overhead of using a general purpose solver must be paid. The model has to be built and solved many times - in some cases building the model is more expensive than solving it. For practical reasons, the number of generated candidates is limited or discretized, and therefore the model is unable to gain a fine-grained control of the pullouts.

Finally, we mention, that this approach assumes that only one line can be pulled out simultaneously. The model should ideally consider the possibility of pulling multiple lines on a particular track; however, given the limited number of lines allowed simultaneously in the bowl, this did not seem to be a critical concern. The proposed model can be extended without much difficulty to handle multiple lines.

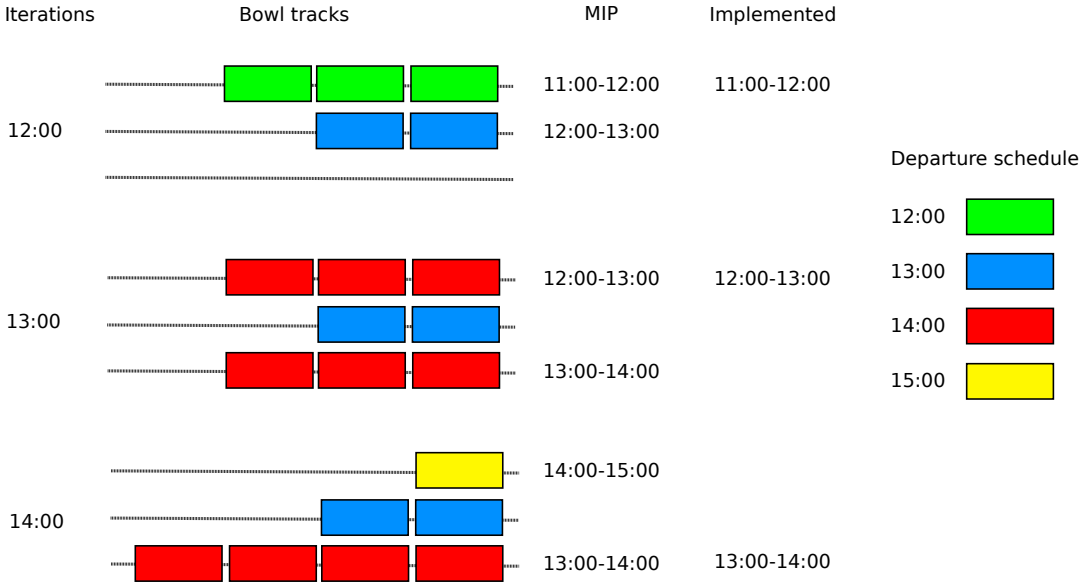


Figure 9.5: An illustration of three pullout iterations in the solution framework. The bowl contains three classification tracks. The boxes represent rail cars where the colors indicates the target departure (block). Here, the iterations occur every hour which means that more units have been processed over the hump in subsequent iterations. The *MIP* column shows the pullout decisions made in each iteration, and the *Implemented* column shows which pullouts were implemented. Finally to the right, the departure schedule is listed. In this example, three pullouts are performed. At 11:00-12:00 and 12:00-13:00 the top track is pulled out with respectively three green and red rail cars. At 13:00-14:00 the bottom track is pulled out with four red rail cars.

9.5 Computational Results

In this section we discuss and benchmark our algorithm by incrementally enabling the more advanced features. For each experiment we report a few key performance indicators. All runs are performed multiple times in order to examine the expected average performance and expected deviation; the variation is due to the heuristic used for the BTAP. We consider the data instances for the HYBA problem introduced by in the RAS Problem Solving Competition in 2014 [19]. Tables 9.1-9.3 respectively summarize the instances characteristics, differences and parameters.

First, we evaluate the framework performance using the *Greedy* pullout algorithm processing inbound trains on the hump by arrival order. The results are shown in Table 9.4. The results show that we can solve the instances within 2 seconds achieving a optimality bound of $30 - 35\%$ for the first two instances, but around a staggering 185% for the third instance. The deviation between runs is very low for the first two (less than 0.1 standard deviation in hours); however, it spans $140.75\% - 287.68\%$ for the last instance.

We do not see much improvement nor loss when activating low or high volume track selection for the BTAP, metioned in Section 9.4.3. Runtime remains unchanged, but a benchmark showed that only the last data set is improved up to 2.4 average dwell hours.

The biggest improvement was observed when using the MIP method for the PAP instead of *Greedy*. The results are summarized in Table 9.5. The benchmark reports a significant improvement in average/bound for all instances. The last instance is, however, still around ten percentage points above the other two. As expected, the runtime is increased, from a few seconds to a few minutes. A noticeable increase in maximum dwell time can be observed compared to the previous benchmark, but the average maximum number of lines is not changed significantly. This is to be expected since the MIP pullout approach does not process the bowl on a first-come first-served fashion.

Inbound trains	702	
Rail cars	52,247	
Rail cars length	3,049,094	feet
Horizon	42	days
Min train length	55.0	feet
Max train length	9,200.0	feet
Average train length	4,343.4	feet
Min train cars	1	
Max train cars	161	
Average train cars	74.4	
Min distinct blocks	1	
Max distinct blocks	32	
Average distinct blocks	15.3	

Table 9.1: Statistics on the instance inbound trains.

Instance No	Tracks	Total Length
1	58	122,489
2	50	109,853
3	42	96,354

Table 9.2: List of data instances. The columns show the instance number, number of classification tracks in the bowl, and the total length (in feet) of tracks.

Hump engines	2	
Pullout engines	3	
Time between train humping	20	Minutes
Humping rate	30	Seconds
Pullout time per job	20	Minutes
Start build time offset	4	Hours

Table 9.3: List of instance parameters.

		Dwell (hours)						
No	Time	Arrival	Bowl	Departure	Total	Max	Lines	Gap
1.0	1.7	0.59	12.35	3.43	16.38	94.7	61.0	30.2%
2.0	1.4	0.92	12.52	3.47	16.90	87.8	53.5	34.4%
3.0	1.2	19.65	12.69	3.51	35.85	91.4	44.8	185.1%

Table 9.4: Results using a greedy humping and pullout strategy. The columns respectively show the instance number, average runtime in seconds, different average dwell time averages in hours, the average maximum no of concurrent lines in the bowl, and finally the relative optimality gap.

		Dwell (hours)						
No	Time	Arrival	Bowl	Departure	Total	Max	Lines	Gap
1.0	287	0.56	11.39	1.22	13.17	129.9	61.0	4.74%
2.0	273	0.57	11.29	1.35	13.21	117.6	54.2	5.07%
3.0	338	1.42	10.92	1.99	14.33	76.6	45.5	13.92%

Table 9.5: Results using a greedy humping method and the MIP pullout method. The columns respectively show the instance number, average runtime in seconds, different average dwell time in hours, the average maximum no of concurrent lines in the bowl, and finally the relative optimality gap.

	Time (s)			Dwell (hours)							
No	Total	Hump	Pullout	Arrival	Bowl	Departure	Total	Max	Lines	Gap (%)	Gap
1.0	522	238	280	0.66	11.25	1.19	13.09	137.53	61.1	4.09%	0.51
2.0	507	238	266	0.67	11.12	1.33	13.12	133.77	54.3	4.37%	0.55
3.0	570	237	330	1.50	10.82	1.95	14.27	73.32	45.3	13.47%	1.69

Table 9.6: Final results obtained using the MIP method for both hump sequencing and pullouts. The columns respectively show the instance number, average runtime in seconds, different average dwell time in hours, the average maximum number of concurrent lines in the bowl, and finally the relative and absolute gap to our lower bounds.

Finally we benchmark the performance of using the PAP MIP together with the HSP MIP method. The results are shown in Table 9.6. A consistent improvement of 0.06 to 0.09 average hours was observed, i.e., roughly 5000 dwell hours. It is observed that this setting improves the average dwell time at the bowl and departure yards, while achieving a slightly increased dwell time at the arrival yard. The HSP MIP method finds a better processing order of the arriving trains, at the cost of processing them more quickly by arrival order. This improvement does, however, come at the cost of approximately twice the runtime. The maximum dwell time and average maximum line usage show no noticeable change. A histogram showing the rail car dwell hour distribution, of one solution, is plotted in Figure 9.6. The majority of the rail cars leave the yard within 25 hours of arrival. Very few rail cars stay in the yard more than 50 hours. The solutions of all instances show a similar shape to Figure 9.6. Compared to the first two instances, the last instance contains fewer rail cars with 0-5 hours dwell time and an increased number of rail cars with more than 25 hours dwell time.

9.5.1 What-If Scenarios

We perform a second line of what-if scenarios in order to identify the current bottlenecks of the data sets. In isolation we change the parameters and data instances in order to achieve different results. All changes are based on the first original instance, i.e, the least restrictive instance with most classification tracks.

The complete set of scenarios and results are listed in Table 9.7. In the first scenario, we increase the size of all tracks in the bowl. In the second we add more pullout engines. In the third we allow a very high number of lines. In the fourth we allow longer outbound trains.

The experiments show that the average dwell is improved most by adding longer tracks or longer outbound trains. Adding more pullout engines or allowing more concurrent lines in the bowl does not have a significant impact.

	Runtime (s)						
Scenario	Total	Hump	Pullout	Dwell	Max	Lines	Change
Long tracks	414.2	247.7	150.9	12.96	71.0	45.0	-0.14
Many engines	662.0	245.5	413.0	13.08	138.2	60.9	-0.02
Many Lines	608.7	255.0	349.6	13.10	138.6	61.1	-0.00
Long trains	429.4	150.2	275.9	12.47	47.6	57.8	-0.63

Table 9.7: Results for the what-if scenario benchmarks. The columns respectively show the scenario name, average runtime in seconds, average dwell time in hours, maximum dwell time of an individual car, the average maximum number of concurrent lines in the bowl, and finally the change in average dwell-time compared to the normal instance.

9.5.2 Delayed Departures

The MIP-based lower bound method from Section 9.4.1 can without much difficulty be extended to allow departure delays. In order to make the altered model tractable, we make alterations that allow an outbound train to select between departure times in a discrete set. The optimal solution is therefore a guess (or an optimistic guess) of the average dwell time when allowing departures.

Due to the discretization, this is not a true lower bound when considering delays; however, it is a good indication that one can expect improvements by considering delays for outbound trains. In our test we allow delays of up to 380 minutes with 20 minutes discretization. This results in a lower bound of 10.10 average dwell hours. This significant reduction suggests that large savings can be achieved by allowing departures to be delayed. We note that roughly half of all outbound trains were delayed in this lower bound solution.

Finally, we benchmarked the result of allowing up to 2 hours delay in our solution methods, thus getting a real solution instead of a bound. These settings generate average dwell hours of 12.31, 12.33, and 12.42 the three datasets. Again, a significant improvement, especially for the last instance.

9.6 Conclusions

In this paper we consider the HYBA problem. We propose a heuristic framework which decomposes the problem into three interdependent subproblems. A version of the algorithm in which we consider greedy strategies for the humping and pullout process obtains acceptable solutions within two seconds. A second version, in which the humping and pullout strategies are solved using MIP models, obtains solutions that are significantly better; however, it does take substantially more time. The runtime is, however, still very reasonable considering the length of the planning horizon. The solutions obtained have a proven optimality gap of a few percent. An additional experiment shows that significant improvements can be obtained by allowing outbound trains to be delayed.

In addition to solving the HYBA problem the proposed heuristic method can be used to estimate the effect of infrastructure or equipment investments. Several what-if scenarios are considered and the results show that the studied data instance can benefit from longer outbound trains and additional track-length in the classification yard. However, allowing more concurrent lines in the bowl or using additional pullout engines does not make a significant difference.

Simple methods have been proposed for finding lower bounds for the problem. The results show that the lower bounds give good estimates for the two first instances. The methods do not take

the bowl tracks into account, which explains why a weaker bound is achieved for the last instance. Promising directions for future research include strengthening this lower bound calculation to reflect the limitations of fewer bowl tracks as well as integrating certain components of the algorithm. In particular, one idea could be to allocate a set of rail cars to bowl tracks when humping a specific car, as opposed to the current approach of greedily allocating each individual rail car a track when it is being humped. Finally, having the ability to dynamically adjust the humping sequence may also yield further improvments.

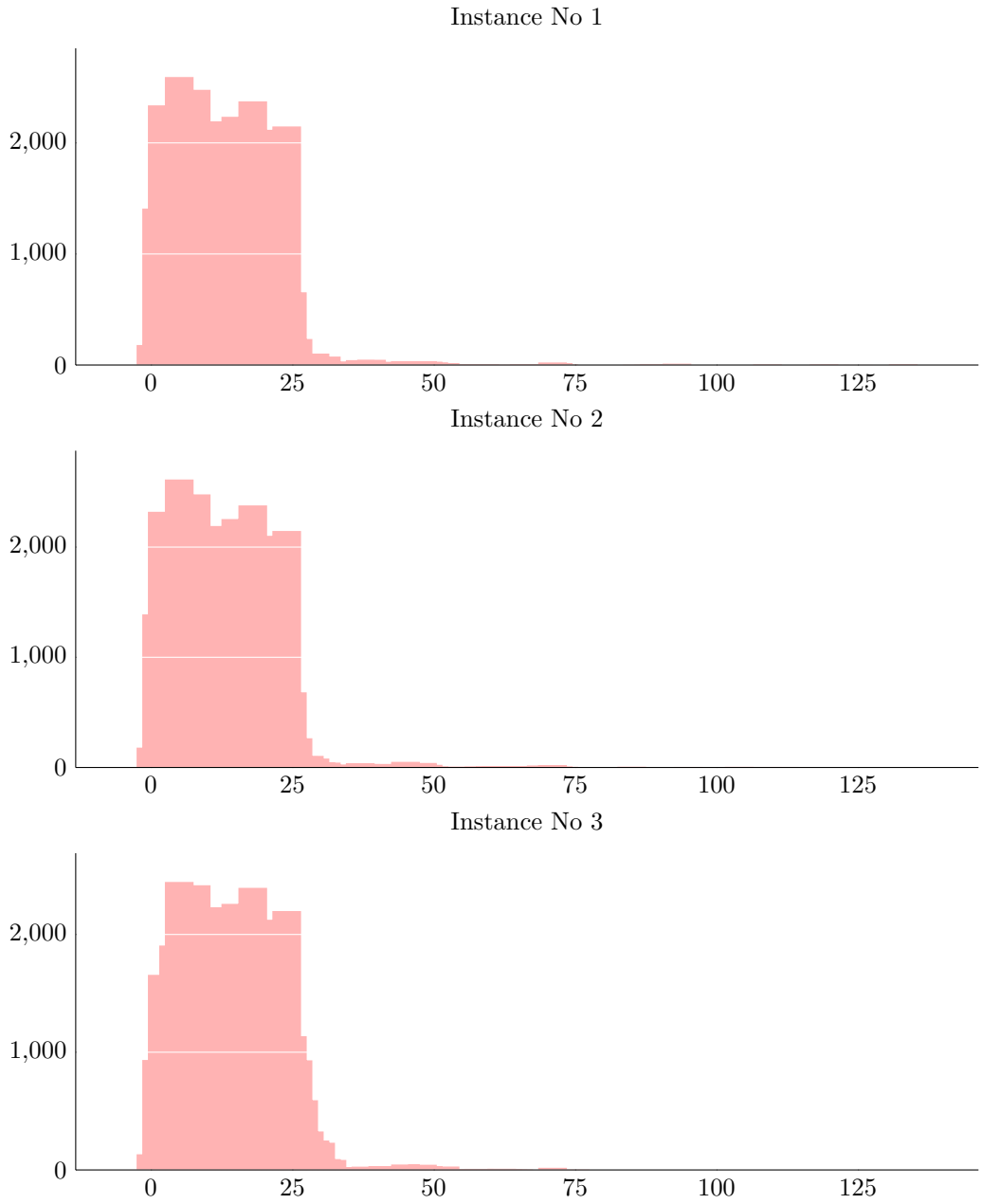


Figure 9.6: Histogram of rail car dwell times. Histogram shows the number of rail cars grouped by dwell hours.

Bibliography

[1] Ravindra K. Ahuja, Krishna C. Jha, and Jian Liu. Solving real-life railroad blocking problems. *Interfaces*, 37(5):404–419, September 2007.

[2] Lawrence D. Bodin, Bruce L. Golden, Allan D. Schuster, and William Romig. A model for the blocking of trains. *Transportation Research Part B: Methodological*, 14(1–2):115 – 120, 1980.

[3] Markus Bohlin, Florian Dahms, Holger Flier, and Sara Gestrelius. Optimal freight train classification using column generation. In Daniel Delling and Leo Liberti, editors, *ATMOS*, volume 25 of *OASICS*, pages 10–22. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[4] Markus Bohlin, Florian Dahms, and Sara Gestrelius. Optimisation of simultaneous train formation and car sorting at marshalling yards. repORt 2013–013, Operations Research, RWTH Aachen University, Jul 2013.

[5] Markus Bohlin, Sara Gestrelius, Florian Dahms, Matúš Mihalák, and Holger Flier. Optimized shunting with mixed-usage tracks. Technical Report 2013-12-19, Swedish Institute of Computer Science, 2013.

[6] Nils Boysen, Malte Fliedner, Florian Jaehn, and Erwin Pesch. Shunting yard operations: Theoretical aspects and applications. *European Journal of Operational Research*, 220(1):1 – 14, 2012.

[7] D. Briskorn and F. Jaehn. A note on "multistage methods for freight train classification". *Networks*, 62(1):80–81, 2013.

[8] Elias Dahlhaus, Peter Horak, Mirka Miller, and Joseph F. Ryan. The train marshalling problem. *Discrete Applied Mathematics*, 103(1–3):41 – 54, 2000.

[9] Jeremiah R. Dirnberger and Christopher P. L. Barkan. Lean railroading for improving railroad classification terminal performance: Bottleneck management methods. *Transportation Research Record: Journal of the Transportation Research Board*, 1995(1):52 – 61, 2007.

[10] Michael Gatto, Jens Maue, Matúš Mihalák, and Peter Widmayer. Shunting for dummies: An introductory algorithmic survey. In Ravindra Ahuja, Rolf Möhring, and Christos Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 310–337. Springer, 2009.

[11] Shiwei He, Rui Songa, and Sohai S. Chaudhry. An integrated dispatching model for rail yards operations. *Computers & Operations Research*, 30:939 – 966, 2003.

[12] Riko Jacob, Peter Marton, Jens Maue, and Marc Nunkesser. Multistage methods for freight train classification. In *Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'07)*, pages 158–174. Schloss Dagstuhl, Internationales Begegnungszentrum für Informatik, Nov 2007.

[13] Edwin R. Kraft. Priority-based classification for improving connection reliability in railroad yards. part I of II: Integration with car scheduling. *Transportation Research Forum*, pages 93–105, 2002.

[14] Edwin R. Kraft. Priority-based classification for improving connection reliability in railroad yards. part II of II: Dynamic block to track assignment. *Transportation Research Forum*, pages 107–119, 2002.

[15] Edward Lin and Clark Cheng. Yardsim: A rail yard simulation framework and its implementation in a major railroad in the u.s. In Ann Dunkin, Ricki G. Ingalls, Enver Yücesan, Manuel D. Rossetti, Ray Hill, and Björn Johansson, editors, *Winter Simulation Conference*, pages 2532–2541. WSC, 2009.

[16] Edward Lin and Clark Cheng. Simulation and analysis of railroad hump yards in north america. In S. Jain, Roy R. Creasey Jr., Jan Himmelspach, K. Preston White, and Michael C. Fu, editors, *Winter Simulation Conference*, pages 3715–3723. WSC, 2011.

[17] Jens Maue. *On the Problem of Sorting Railway Freight Cars. An Algorithmic Perspective.* PhD thesis, ETH Zurich, Göttingen, Germany, July 2011.

[18] Harry N. Newton, Cynthia Barnhart, and Pamela H. Vance. Constructing railroad blocking plans to minimize handling costs. *Transportation Science*, 32(4):330–345, April 1998.

[19] Institute for Operations Research Railway Applications Section and the Management Sciences (INFORMS). Railroad hump yard block-to-track assignment, 2014.

Part IV

Train Speed Profile Optimization

Chapter 10

A Dynamic Programming Approach for Optimizing Energy-Efficient Velocity Profiles with Speed Limitations and Passage Points

Jørgen Thorlund Haahr* David Pisinger* Mohammad Sabbaghian†

*Department of Management Engineering, Technical University of Denmark,
Produktionstorvet, Building 426, DK-2800 Kgs. Lyngby, Denmark
jhaa@dtu.dk, dapi@dtu.dk

†Cubris ApS
Ebertsgade 2, DK-2300 Copenhagen S, Denmark
m.sabbaghian@cubris.dk

[1] **Abstract** This paper considers a novel solution method to generate optimized train speed profiles. The solution method is based on a Dynamic Programming (DP) methodology that is different from existing methods in literature using DP. Instead of using uniform discretization of space, time or speed, we rely on an event-based decomposition that drastically reduces the search space. Compared to other approaches, we include passage point constraints, e.g., mimicking speed signals. In collaboration with the technology company, Cubris, we compare the solution method with a state-of-art solver. Based on an extensive number of real-life problem instances our benchmark shows that the proposed solution method is able to improve existing solutions by 3.3% on average. The computational times are very low, making it possible to handle unexpected changes in speed restrictions or timings by recalculating the schedule. This is a great advantage over static offline lookup tables. Selected details of the method and benchmark are kept confidential for sensitivity reasons.

[1]Submitted to *Transportation Science*, November 2015

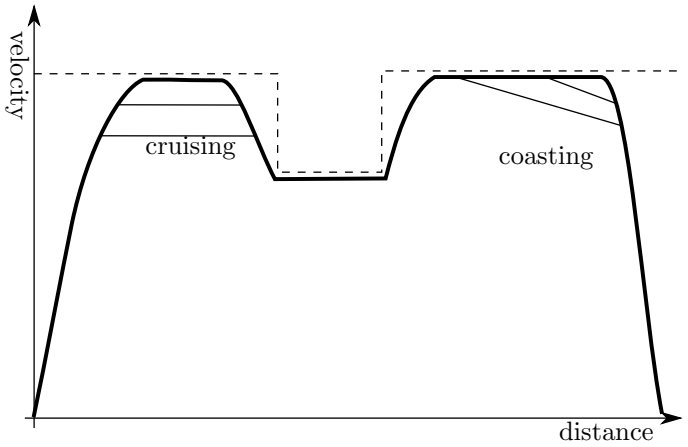


Figure 10.1: The plot shows examples of feasible speed profiles respecting the speed-limits, which in turn are drawn using dashed lines. The bold path illustrates an aggressive strategy following the speed-limits as closely as possible by using full acceleration and braking, thus arriving as soon as possible. Alternative and more energy efficient profiles can be achieved, at the expense of a longer trip time, by lower cruise speed or by performing coasting as shown. The available travel time, $T_{\max}$, limits the number of feasible profiles.

10.1 Introduction

Railway operation provides a fast and efficient mode of transportation, but despite being very energy efficient compared to most other transportation forms, the overall energy consumption is large. Three of the largest railway network operators in Europe spend up to €1.75 billion annually on energy [2]. In Germany, for example, railway traffic consumes 2% of the country's energy usage. Thus, operators have strong incentives to reduce energy usage while maintaining punctuality. Passenger train services are often operated daily in tight preplanned schedules while a number of long and heavy freight trains are planned in a more *ad hoc* basis. Even minor energy reductions in individual train schedules can lead to significant savings. Studies have shown that potential savings of $5-20\%$ are possible by optimizing train speed profiles [10].

Regardless of whether it is a freight or passenger train service, the train travels from one station to another, possibly making intermediate stops. Timeslots on tracks are allocated to the train allowing passage through the rail infrastructure, and strict timing is imposed on arrivals and departures to enforce customer satisfaction and a high network utilization. A train driver thus knows in advance when he has to depart from the current station and arrive on the next.

Given a maximum trip time $T_{\max}$ the speed profile, i.e., how fast the train is driving throughout the trip, is mainly determined by the train driver. This is not a trivial task, so in order to avoid delays the driver often decides to drive as fast as possible or stay ahead of time by following the speed-restrictions as closely as possible thereby braking and accelerating aggressively. Apart from needless waste of energy, the fastest speed profile is also in some cases infeasible due to passage point restrictions, e.g. due to the interlocking system for track-changes or other conflicts. Examples of different speed profiles are illustrated in Figure 10.1. Essentially, a speed profile consists of up to four different actions: acceleration, braking, cruising and coasting. Note, that the first two actions can be performed between with $0-100\%$ effect, where braking may regenerate a portion of the energy.

Energy usage corresponds to the fuel or electricity consumption of the train and is often measured by the traction effort of the train over some trip. A train driver directly controls the consumption by regulating the speed throttle. In order to save energy the train driver can operate the train at lower speeds, thus loosing less energy to natural resistances, or perform coasting. There is a

clear trade-off between trip duration and energy consumption, in general, shorter trips require more energy. Analytical formulas can project the effect of increasing or reducing speed, however combining a complete profile is not trivial. Multiple additional constraints further complicate the problem, e.g., changing speed restrictions, passage point time requirements and changes in altitude. We will refer to this optimization problem as the Train Speed Profile Problem (TSPP).

In this paper we propose a novel Dynamic Programming (DP) methodology for optimizing the TSPP in real-time that is suitable for an on-board Driver Advisory System (DAS). The method is in some sense elegant and does not rely on advanced software packages or mathematical solvers. In addition to gradient forces and speed-restrictions the framework can also handle passage points. To the best of our knowledge a solution method for (multiple) passage points has not been proposed in literature before. The solution method is based on a space-velocity graph representation of partial speed profiles that are computed using various equations for train motion. Compared to previous approaches we do not uniformly discretize space nor time. An advantage of this representation is the ability to extend the number of possible choices without changing the underlying solution method. The approach is not guaranteed to be optimal but, with the inclusion of intelligent partial profiles, this method is able to find high quality solutions. The choices of the underlying graph are optimized using a label setting algorithm that minimizes energy usage subject to a time resource constraint.

We benchmark the method on multiple real-life instances from different train operators. The instances include multiple actual speed-restrictions, passage points and altitude profiles. We show that the computational times are sufficiently small for producing continuous real-time speed-advice. The results are compared to a fast heuristic method, which we refer to as the *C-heuristic*. For the purpose of this study, we compare both approaches in a static environment between two train stops.

In this work some important assumptions are made. Firstly, we assume that the train is able to hold speed (maintaining a constant velocity), e.g., using cruise control. Even if this is not an option, we assume that a post-processing step can be performed that approximates a constant speed by switching between acceleration and coasting. Secondly, we assume non-steep track slopes, i.e., the train has sufficient engine and braking power to be able to hold speed at any point in time. We note however, that the given framework can without much difficulty be extended to include steep hills. Thirdly, we assume that only a discrete set of cruising speeds can be used, since it is impractical to give speed advices of non-rounded numbers to a train driver.

10.1.1 Related Literature

Much research is conducted in the area of determining optimized speed profiles. Theoretical results were established already a few decades ago [3, 11, 19], and methods to optimize speed profiles are proposed by many authors in different settings. In the following we briefly mention some fundamental theoretical work and the related work on numerical solution methods. We refer to [10] for an introduction to train running time estimation and energy-efficient train operation. A recent review of both analytical and numerical approaches for determining optimal speed profiles is presented by Want et al. [20].

Fundamental theoretical work is established by Howlett and Pudney [13]. In [12] they show that an optimal driving strategy must have certain properties. With continuous speed control they show, using the Pontryagin principle, that the optimal speed profile consists of a power-hold-coast-brake strategy. The simplified optimization problem thus consists of identifying the optimal switching points. However, with multiple speed restrictions and steep hills this strategy must contain multiple power-hold-coast-brake sequences. Cheng et al. [5] study analytical approaches for handling multiple speed limits with continuous speed. Although there is a lack of theoretical work on this subject, the significance of passagepoints is clear as many railway operators have fixed time-allocations of infrastructure blocks.

Franke et al. [8] propose a DP approach for solving the TSPP. They model multiple speed restrictions, regenerative braking and gravitational forces on slopes. In addition, they consider a more realistic modelling of engine efficiency and power losses in the propulsion system. They do, however, not model passage points. Implementational and benchmarking details of the DP algorithm are omitted but they show that the approach provides significant gains over very simple strategies in one studied test case.

A short study of another DP programming approach is presented by Ko et al. [16] for solving a variation of the TSPP. They include complicating constraints that provide a more realistic model of the electric motive force and regenerative braking. Time is discretized uniformly in the approach and they show one benchmark with multiple speed restriction and track slopes. A solution is found within 22 seconds for the study and they conclude that the approach is acceptable for practical purposes.

In addition to an Ant Colony Optimization (ACO) and a Genetic Algorithm (GA) approach Lu et al. [18] also consider a DP approach. The solution space is heuristically restricted in order to reduce the large state-space and thereby improving both computational time and memory usage. A lattice grid is adopted to reduce the feasible region, and choices that deviate too much from an average speed are pruned. The methods are benchmarked against a single instance using three different journey times. Multiple speed restrictions and altitude data are included. The DP approach produces the best results, but the computational time requirement is close to one hour.

Another DP approach is proposed and tested by Larranaga et al. [17]. They consider extensions to the problem using more realistic curves of the tractive and brake efforts. Short journeys of a metro system are used as a case study. In the DP method, time, space and velocity are discretized uniformly and the solution method is able to solve a short trip within one hour of computational time. All states are computed and recorded which means that the results can be used as a lookup table which can be used if the actual speed profile deviates from the optimized one. They conclude that a fine discretization is needed in order to obtain accurate results.

Compared to previous work, we include passage-points as additional constraints and aim for computational times usable in a real-time system. The DP presented in this work does not discretize space nor time uniformly and states are only evaluated if they can lead to an optimal solution.

Several heuristic methods for solving the TSPP have been presented in literature. We briefly mention some of the most important studies here. Jong et al. [15] present a heuristic based on a logic flow-diagram used to identify promising solutions. They compare it to a commercial software package using realistic train journeys provided by the Taiwan Railway Administration. Computational times and solution quality are competitive. Wong et al. [22] consider the TSPP without the possibility to cruise at a certain speed. They present a few methods for solving the problem by determining either one or multiple coasting points, i.e., with or without speed restrictions. One short and one long trip instance are considered where the solution methods require respectively around 90 and 350 seconds to solve. Chang and Sim [4] present a GA for producing a driving profile lookup table for an automated train operation controller. Multiple speed restrictions are included and a more detailed model for regenerative braking is used. No computational times are reported for the one considered instance. A GA approach has been considered in many instances in literature [9, 21, 22].

The structure of this paper is as follows. In Section 10.2, we present a problem definition for the train speed profile problem and outline important assumptions. Then, we present a graph representation of the problem in Section 10.3 which is the foundation of our solution solver. We show how to construct the problem graph using partial profiles of acceleration, braking, cruising and coasting. The dynamic programming solver is then presented in Section 10.4, where we discuss and present an outline of the algorithm used to solve the graph problem instances. Finally, results are presented in Section 10.5 and our conclusions are summarized in Section 10.6.

10.2 Problem Definition

Assume that the considered train, according to a given timetable, has to depart from one station and arrive at the next station at a specific time. The initial and final speed is equal to zero. The train is assumed to depart at time 0, and a maximum total trip time, $T_{\max} \geq 0$, is given (possibly including some buffer time). The distance to travel is determined by the track-layout between the stations and consist of changing track slopes. Although not necessary, assume that the altitude profile consist of piece-wise linear slopes. Different speed restrictions may exist on the trip, i.e., the whole trip is partitionable into a consecutive sequence of segments with changing speed limits. Assume that no lower bound is imposed on the speed limit, although it is easy to extend the model to handle this case. If the velocity of the train exceeds the upper limit, at any point in time, then that profile is infeasible. A train must brake in advance before reaching a new segment if the speed limit there is lower than the current velocity.

Assume that the train has continuous speed throttle control, and the ability to hold a certain velocity once it has been reached. Trains with discrete controls are a different line of study, but we note that a hold-speed (i.e. cruising speed) phase can be approximated in a post-processing step. Assume, in this work, that no steep hill exists on the trip that makes it impossible to hold a certain velocity. Note, however, that the solution method we propose can, without much difficulty, be extended to include such. The characteristics of the train are known, such as the total mass, acceleration power, service braking capability and resistance parameters. Although not a limiting constraint, assume that no regenerative braking can occur. Regenerating power while braking the train has been technically possible for some time, but how to efficiently use that power is non-trivial when optimizing one train speed profile in isolation.

Multiple passage points can exists on the trip. A passage point consists of a time-window and a location. The train can only pass the location in the specified time-window. No additional speed restrictions, other than the governing ones, are enforced. It is for example not required for a train to come to a full stop (at the passage point). This is a constraint that e.g. mimics a signal light where the train can only pass while the signal is green.

Any speed profile that respects the mentioned constraints is feasible. The optimal speed profile is defined to be the feasible profile that consumes the least energy. Other interesting solution characteristics, not considered in the paper, could be included such as a *punctual* arrival or the overall robustness of the profile. Passage time windows, especially the final destination, state that any speed profile that arrives within a certain threshold of the planned time is feasible; however, it may be more attractive to arrival more punctually than saving a small amount of energy. We define the fastest speed profile to be the profile that runs the trip with minimal time usage. This profile is unique and can be generated by accelerating the train as fast as possible to the active speed limit (or the maximum train speed), only braking just in time for a speed limit. Figure 10.2 shows an example of the TSPP.

A solution method for the defined problem can be used as a responsive on-board DAS, provided that solutions can be found very quickly. Such real-time systems are extremely useful for coping with variations in the daily schedules (e.g. cargo/passenger weight or climate) as well as larger disturbances (e.g. changed passage timings and trip times). For the sake of simplicity, we present the problem of optimizing from a full stop to the next stop, but with slight modification this assumption can be lifted in the solution method.

There are many arguments in favor of a real time onboard algorithm, as opposed to pre-calculated speed profiles. The most important reason is the inherent variation in the process. There are many reasons why the actual journey is different from the planned journey; examples are temporary speed restrictions, emergency speed restrictions, route or material changes, and train defects (reduced power). Especially, a Traffic Management System, which solves disruptions by retiming and rerouting trains in real time, will make the speed profile prone to last minute changes. The reason to calculate the advice onboard is to minimize the length of the control-loop[6]. This is the

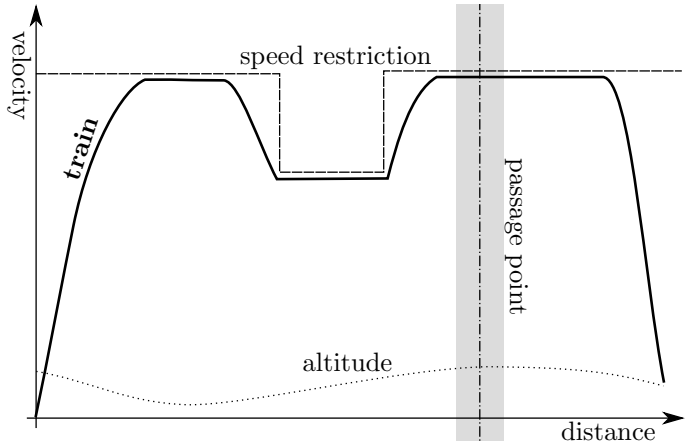


Figure 10.2: An example of a problem instance. The figure shows a distance and velocity plot for a feasible train profile. Governing speed limits are illustrated as dashed lines, and the altitude profile is shown with a dotted line. The passage points requires the train to pass through a certain point within a certain time window. The bold black path depicts the fastest speed profile.

time from detecting the deviation (from the time path) and the actual change of the driver action, i.e., receive a GPS signal, send to central, compare time, recalculate, send back, update screen, driver reaction time and etc. If this loop is too long, the advice has already become obsolete and recalculation is necessary. Calculating the advice onboard and within a second allows the driver the most possible time to adjust the train speed according to the calculated speed profile.

10.3 Graph Representation

The proposed solution method is essentially a general framework that consists of three main ingredients, or modules: formulas to model the train dynamics, a graph representation, and a solver. Note that any module can be replaced with no or minor changes to the overall framework. In the following we explain the individual components.

10.3.1 The Train Dynamics

Modelling train dynamics such as acceleration curves and resistances is necessary in order to project a speed profile. The accuracy depends on the chosen assumptions and simplifications. A balance between the achievable computational time and accuracy is needed. In the following we describe the formulas used in this paper.

As mentioned earlier, the optimal speed profile for a single trip consists of a power-hold-coast-brake strategy [13]. The acceleration (i.e. power) and braking phases thus consist of full acceleration capacity and maximal allowed braking. Intuitively, doing either at partial capacity only results in loss of trip buffer time. Assume that the motion of the train can be reduced to the motion of a point mass [13]. Long trains with distributed mass are reducible to a point mass by modifying the altitude profile accordingly.

In the coasting phase no energy is applied to accelerate the train, thus at a certain speed only vehicle resistances such as friction, mechanical and air affect the train speed. We adopt a quadratic formula, that has been used for decades, for modelling such resistances:

$$R(v) = A + B \cdot v + C \cdot v^2$$

The coefficients A, B and C are train specific. Better known as the Davis Equation[7], it states that the resistance is speed-dependent modelled by a quadratic function. In the coasting phase, this equation can be used to describe how much kinetic energy is lost over time or distance to resistances. Given a certain starting or ending speed, it is possible to derive how fast the velocity changes, or when the train reaches a certain target velocity. In addition to the gravitational pull, this equation models the coasting phase.

Gravitational pull is determined by the track slope:

$$F_{grav} = m \cdot g \cdot \sin(\alpha)$$

Where m is the train mass, $g \approx 9.8^{m}/_{s^2}$ is the gravitational acceleration and α is the angle of inclination.

Thus the following determines the net force during coasting:

$$F_{coast} = R(v) + F_{grav}$$

During the hold phase, speed remains constant. Energy must be applied by the engine in order to overcome train resistance and gravitational pull. In case of a downward slope, it may not be necessary to apply any energy at all, and potentially requires initiating partial braking.

In the braking phase, we assume that the brakes can apply a constant force. Due to reasons such as safety, equipment tear-and-wear and passenger comfort a fraction of the full braking capacity is adopted, i.e., service braking F_{sb}.

$$F_{brake} = F_{sb} + R(v) + F_{grav}$$

Acceleration over time or distance is approximated using the following equation:

$$F_{acc} = \min\left(\frac{F_{engine}}{v}, F_{amax}\right) + R(v) + F_{grav}$$

Where F_{engine} is the maximum engine tractive force. At low speeds $\frac{F_{engine}}{v}$ has proven to be inaccurate [13] - due to physical factors such as adhesion and overheating. F_{amax} denotes the maximal acceleration that is possible for the train at low speeds. Acceleration is naturally also affected by train resistances and gravity.

In this paper, we assume that energy consumption (in terms of fuel or electricity) is proportional to the mechanical energy consumed. Thus, calculating the energy usage is a matter of accumulating the work required by all acceleration and cruising phases. If we considered regenerative braking, the deceleration force would reproduce some of that energy. Although not difficult to include, this aspect is not pursued in this work.

In the following section, the formulas above will be used to generate partial profiles, and it is central to find intersections between these partial profile. This can be done analytically using the above formulas or estimated numerically. We note that the analytical approach quickly becomes impractical as cumbersome integration is required to find eligible expressions. Furthermore, some form of numerical search (e.g., Newton's method) is required to solve the roots.

We assume the altitude profile is piecewise linear, i.e., the whole trip is partitioned into segments, each having a constant track slope. A trip is thus split into segments defined by $s_0, s_1 \dots s_n$ split points. This assumption simplifies the above formulas and provides reasonable accuracy. Accuracy is adjustable by adding more split points.

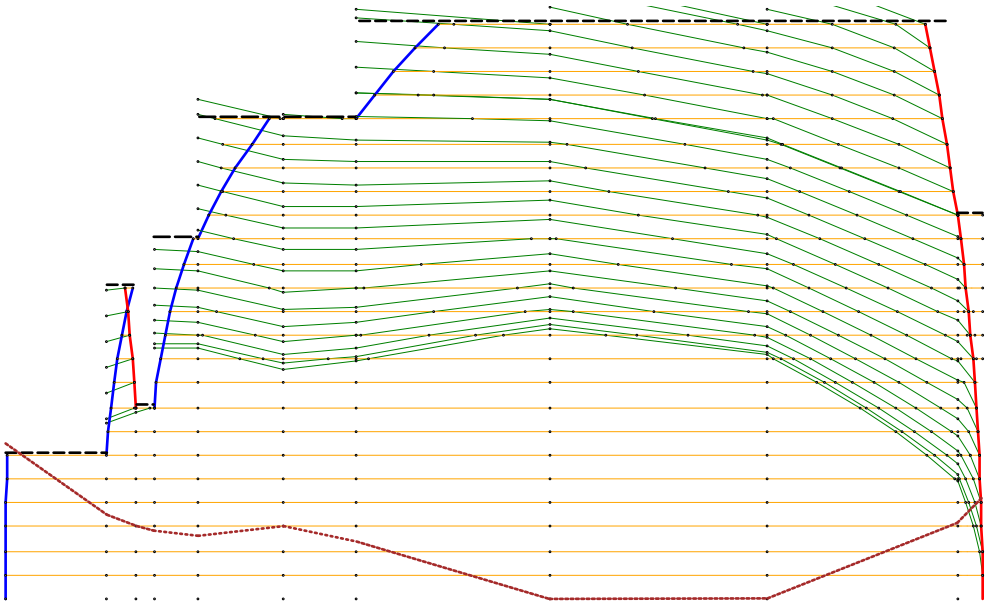


Figure 10.3: An example of a generated speed profile graph. Speed limits are depicted using solid black dashed lines, and the altitude profile using brown dotted lines. The other lines, which are edges in the graph, depict the partial acceleration(blue), braking (red), cruising (yellow), and coasting (green) profiles.

10.3.2 Speed Profile Graph

The formulas presented in Section 10.3.1 can trace partial acceleration, braking, cruising or coasting profiles in a velocity by distance plot. We now present a graph that is generated based on such partial paths where the intersection of these define the vertices of the graph. An infinite number of partial paths exist, but we select a subset of these, which are most likely to appear in an optimized speed profile.

Figure 10.3 shows an example of the described graph. From the initial position project acceleration profiles and whenever the speed limit is increased do the same. Project braking profiles from the final destination and from decreasing speed restrictions. From braking curves project coasting profiles backwards. Finally, cruising profiles are inserted at the predetermined discretized set of velocities. The altitude profile is shown as a dotted line; observe the gravitational pull on the coasting trajectories. Note, for the sake of simplicity, we add vertices at the border of each region. Vertices with only one edges, or only one *in* and *out* edge can be removed in a preprocessing step. Performing further preprocessing reduces the graph, leading to an average reduction of roughly 70%.

A central concept in the graph is the *region*. A trip is first partitioned into a list of consecutive regions, where each region defines a stretch with identical characteristics. This simplifies some of the analytical formulas, but also identifies key positions. The regions are directly related to the split points defined earlier as two split points define the borders of each individual region. In addition to tracks slopes, we extend the set of split points to include speed restrictions. Note, that the regions are not partitioned uniformly by distance (or time) in general. A change in track slope or maximum speed restriction defines a new region.

As mentioned, one important goal is to give drivers rounded speed advice, e.g. 5, 10, 15 or 20 kilometers per hour. Thus, only a predetermined discrete set of velocities will be considered as feasible cruising speeds. The driver will therefore only receive advice to accelerate or brake to one of these selected rounded velocities. Following fractional speed advice is more impractical to realize by the drivers, thus rounded velocities are, in a sense, more robust. Furthermore, rounding

exact speed profiles in a post-procedure is also non-optimal as this will cause the train to arrive at the wrong time if the driver follows the speed advice exactly as instructed. The profile is unstable as drivers will face difficulties to follow it; with use of a DAS this potentially results in several profile recalculations.

With multiple speed limits, it is not trivial to identify the optimal solution nor the partial paths that it consists of. It is however known that the solution has certain properties. Acceleration should be performed as quickly and early as possible in order to save time. Similarly, braking should be performed as quickly and late as possible in order to save time. Intuitively and informally, late acceleration or early braking does not save energy, and thus only results in an increased trip time without energy gains. Acceleration and braking profiles are therefore inserted at region borders whenever the speed limit changes.

Multiple cruising stages may exist, and finding the optimal velocity for each cruising stage is not trivial. As described, only a selected discrete set of velocities is allowed, thus we include these as partial profiles. We note that the speed limit is included in the selected set.

A rule set for coasting is more difficult to determine, which is why we rely on a heuristic approach for choosing when to start coasting. However, from the optimal strategy we only need to consider coasting before a braking stage. For every braking curve we therefore adopt a number of partial coasting profiles that intersect with the braking curve. As a heuristic choice we decided to generate one partial coasting profile that intersects with the braking curve for each of the selected discrete speeds, i.e., at the end of a coasting phase the driver will be told to brake when he hits one of the preselected rounded speeds.

We can consider the constructed graph $G(V, A)$ as a directed graph. Intersections in the graph define the vertices, V, of the graph. The segments of partial paths between two vertices define the arcs, A, of the graph. By construction the graph is acyclic as arcs only connect vertices strictly forward in time. Associated to each vertex $v \in V$ is a position and a speed, and to each arc $a \in A$ is a time and energy consumption. An arc represents a driving choice for a period of time, e.g., accelerating from one position and velocity to another. We define the source, $v_s \in V$, as the initial position at zero velocity, and the sink, $v_t \in V$, as the final destination at zero velocity. We let $outEdges(v) \subseteq A$ denote all arcs originating from $v \in V$.

Any path from v_s to v_t represents a driving profile but the path is not necessarily feasible. Although the actions represent a feasible sequence of choices they will not necessarily respect the time requirements. If the accumulative time of the arcs on the path is less than the total time available, then we say it is a feasible path.

Passage Points

Passage points are a natural extension to the proposed graph structure. Firstly, positions of the passage points are now also splitting points in the graph, resulting in an increase of regions. At these points (on the trip stretch) the train is required to pass in the specified time window. We define the final passage point as the arrival time at the final destination.

In this extension, additional acceleration curves are added such that all of the cruising speeds are reachable before passing the passage point. The additional acceleration curves are needed in the special case where a large excess of time is available before the passage point. In order to save energy a low speed should be selected, however depending on the next region, it might not be feasible. It may not be feasible to pass the next passage point if the next region is entered with a low speed.

The passage point extension naturally imposes additional constraints to a feasible path. In addition to a finding a connected path, all passage points must be reached within a specified time

window.

10.4 Label Setting Algorithm

We use the train speed profile graph in Section 10.3.2 as a basis for finding optimized profiles. Any path from the source to the target vertex represents a feasible sequence of actions, however it is not necessarily feasible as the available time windows may be violated. Therefore, the TSPP problem has to be solved as a Resource Constrained Shortest Path Problem (RCSPP). This is a well-studied problem and we refer to Irnich et al. [14] for a more in-depth description of how to solve this problem.

We propose a label setting algorithm to solve the RCSPP, which essentially is a DP approach. The considered problem structure has an optimal substructure property, but instead of computing a table of all states we are only interested in the optimal solution and therefore can discard inferior states. In contrast to previous DP approaches, we do not generate a lookup table for the whole trip but rely on re-computation when the train deviates from the original plan. We limit this section to a brief description of the label setting algorithm. Note, that using this methodology we find an optimal path in the generated RCSPP, but the solution is only sub-optimal for the TSPP since not all possible transition states are present in the graph.

In the RCSPP, the goal is to minimize the overall energy consumption while respecting the available trip time, $T_{\max}$. The available time can be considered a resource, which is consumed when traversing the arcs in the graph. A label, l, is a sequence of connected arcs in the graph, i.e., a partial path. The energy consumption, $cost(l)$, and time usage, $time(l)$, correspond to the accumulative energy and time usage of the entailed arcs. The source vertex, $v_s \in V$, is the head of any label l and we let $tail(l)$ denote the tail. Note, that for any arc $a \in A$, $cost(a)$, $time(a)$ and $tail(a)$ are defined similarly.

The algorithm is essentially a full enumeration of all possible paths, starting from v_s. The first label simply consists of the source node, having zero cost and no time usage. In a recursive manner, new labels are generated for every possible extension of current label. In the end, all possible partial paths originating at v_s will be generated. However, in order to be more efficient some search branches are pruned and discarded in the process if they prove to be fruitless. We adopt the use of infeasible labels and dominans to efficiently prune the states in the enumeration process.

Theorem 10.1 (Infeasible Label I)**.** *Infeasible labels can be pruned. A label l is infeasible if $time(l) > T_{\max}$.*

Proof. All extensions of an infeasible label will also be infeasible. This follows from the fact that all arcs are assumed to consume a non-negative amount of time, i.e., not moving backwards in time. □

Theorem 10.2 (Infeasible Label II)**.** *Infeasible labels by time bound can be pruned. A label l is infeasible by time bound if $time(l) + \delta > T_{\max}$ where δ is the minimum time required to reach v_t.*

Proof. It is not necessary to look at extensions of the label if no future choice can lead to a feasible profile. □

Theorem 10.3 (Dominated Label)**.** *A dominated label can be pruned. Consider two labels a and b at the same vertex (i.e. having same speed and location). Label l_a dominates label l_b if $cost(l_a) \leq cost(l_b)$ and $time(l_a) \leq time(l_b)$.*

Proof. First, we can assume that both labels are not infeasible. All arcs are assumed to consume a non-negative amount of time as no arc choice will reduce the time usage. Both labels define a

path from the source to v, i.e., they are at the same position and velocity. The possible future extensions of both labels is therefore identical (disregarding the time resource). Since the cost of l_a is no greater than l_b, l_b can never lead to a cheaper future path as the same path is extendable from l_a with no more (i.e. less or equal) cost. In addition since l_a a has consumed no more time than l_b it is able to reach no less (possible more) vertices. Hence, we can safely remove l_b from further consideration without cutting the optimal solution away. □

Algorithm 5

```
1:  procedure LabelSettingAlgorithm(G(V, A), T_max)          ▷ graph and trip time
2:      L ← {CreateLabel(v_s, 0, 0, ∅)}                      ▷ stack of unprocseed labels
3:      B ← ∅                                                ▷ best solution
4:      while |L| ≥ 0 do
5:          l ← Front(L)
6:          L ← PopFront(L)                                  ▷ Remove l from unprocessed list
7:          v ← tail(l)
8:          ApplyDominance(v)                                ▷ pruning dominated vertices
9:          for a ∈ outEdges(v) do                           ▷ performing extensions
10:             v' ← tail(a)
11:             l' ← CreateLabel(v, cost(l) + cost(a), time(l) + time(b), l)
12:             if IsFeasible(l') then
13:                 L ← L ∪ {l'}
14:                 if v' = v_t ∧ cost(l') < cost(B) then
15:                     B ← l'
16:     return B
```

The general label setting algorithm is summarized in Algorithm 5. Starting with the source vertex, labels are processed in iteration by making all possible extensions. Infeasible extensions are pruned from consideration, and dominance is checked in order to prune fruitless labels.

Passage Points Extension

An extension is needed in order to deal with the passage point constraints since the described algorithm does not require labels to pass these locations in the pre-specified time windows.

In Algorithm 5, instead of assigning the accumulated time, $t_{l'}$, on line 11 we assign to the generated label, l', a time consumption of $\max(t_{l'}, t_p)$ where t_p corresponds to the earliest arrival time at passage point p. If no passage point exists at the considered vertex then $t_p = 0$. This modification of the algorithm keeps the structure of the problem intact, and we can use the same pruning rules. In addition, the time bound infeasibility rule is strengthened as it can be applied for the latest arrival at all passage points, not just the final destination and $T_{\max}$.

The modification has one important drawback: the produced solution may be infeasible as the modification allows a label to move forward in time without changing the position or velocity. This is naturally not possible for a train, unless it is standing still. We propose two different ways to accommodate this issue. Firstly, assigning a cost penalty on violations, according to the size of the violated duration, discourages solutions that arrive too early. Secondly, a post-process can repair the infeasibility by adjusting the profile accordingly, such that it arrives later in time but at the earliest time possible at the correct speed. In the experiments we have adopted the latter.

10.5 Computational Results

In this section we benchmark the proposed solution method using real-life instances from various countries. A summary of the considered instances in shown in Table 10.1. The instances are divided into three different classes, each representing distinct trips. The first class (A) consists of intercity trips provided by a confidential railway operator. The last two classes (B and C) are intercity trips provided by the principal train operating company in Denmark (DSB). All instances include multiple speed limits and in many cases also multiple passage points. The altitude profile is not negligible, hence many piecewise linear slopes are used to approximate the track slopes. The arrival time at the final destination is given for each instance, where the input data allows the train to be delayed a few seconds.

The benchmarks are performed using a Intel (R) Core^{TM} i5-3320M CPU @ 2.60GHz processor with 8 gigabytes of main memory. The results are shown in Table 10.2 where the solution method described in this paper (LSA) is compared to a solver (*C-heuristic*) provided by Cubris[1]. LSA and *C-heuristic* method solve the same problem using the same objective, however, a few differences exist. First, *C-heuristic* does not rely on linearized track slopes. Secondly, additional soft constraints, such as minimizing the number of driver actions, are taken into account in *C-heuristic*. The parameter settings of *C-heuristic* have, however, been adjusted to make the comparison as fair as possible. For the sake of comparison, the energy consumption is normalized into watt-hours per ton-kilometer ($^{wh}/_{tk}$), which we will denote as the cost. We measure the results of LSA and *C-heuristic* relative to the cost of the fastest profile (*FP*). The *FP* is the speed profile that uses the minimum travel time possible for a journey. This profile follows the speed restrictions as closely as possible, which helps the train to stay ahead of time (rather than behind the schedule). This speed profile represents the choice made by an energy-inattentive driver. The cost of the *FP* highlights the potential savings of the benchmarked methods.

In general, the results show that the LSA produces high quality solutions within 0–2 seconds. We note that some of the input data can be preprocessed further, thus reducing computational time. A large portion of the time is spent in the label setting algorithm, but time spent preparing the input data and generating the underlying graph is not negligible. An average improvement of 0.3, 1.2 and 0.4 $^{wh}/_{tk}$ in cost (1.5%, 2.6% and 0.8% in savings) is observed for respectively *A*, *B* and *C* instances compared to the *C-heuristic*. Compared to the energy usage of *C-heuristic*, the relative improvement per kilometer is 3.6%, 4.7% and 1.5% respectively. The average improvement for all trips, including all classes, is 3.3%.

Compared to the A instances, relatively high computational times are observed for the B and C instances. This is a natural consequence of the increased distance, the number of regions and the graph size, resulting in a larger number of considered labels.

The LSA consistently arrives as late as possible, which in turn gives more room for cost reductions. In contrast to the *C-heuristic* method where the arrival varies around on the planned time, and even arrives before time in some cases (up to 2 seconds).

The *C-heuristic* finds solutions extremely fast, all measured computational times are less than 100 milliseconds. Significant gains are achieved compared to the *FP*, but the superior results of the LSA suggest that the *C-heuristic* sometimes makes some poor choices. We note that some discrepancy might appear in the results, since the LSA gets a linearized altitude profile as input, while the *C-heuristic* works on a continuous altitude profile.

Comparing the costs of the two algorithms, a few outliers need some explanation. In A07 and C06 the LSA surprisingly finds significantly worse solutions. After inspection of the results, we discovered that this is due to a combination of large segments in the altitude profile and the discrete set of cruising speeds. This considerably restricts the choices of speed for LSA. Splitting the long segments in the altitude profile into smaller parts will reduce this problem.

A relative high difference is observed in a few cases: A05, A16, B00, B02 and C02. After inspecting

Instance	Distance	Speed Limits	Passage Points	Altitude Difference	Track Slopes	Trip Time
A00	6.3	7	1	9	5	390
A01	32.9	9	5	28	13	1 020
A02	37.7	4	3	54	17	1 110
A03	29.8	7	2	48	10	930
A04	28.0	8	2	62	8	1 080
A05	19.8	12	3	43	8	780
A06	14.7	5	1	52	3	570
A07	10.9	2	1	30	4	420
A08	9.8	7	1	51	3	390
A09	7.5	3	1	11	5	300
A10	14.2	3	1	35	4	510
A11	21.3	6	3	84	3	750
A12	16.5	8	2	119	4	660
A13	22.3	7	1	105	6	720
A14	1.1	4	1	20	2	180
B00	50.3	19	10	55	27	1 575
B01	28.9	3	4	20	15	795
B02	23.5	4	2	102	10	765
B03	15.5	4	2	37	4	495
B04	14.8	6	1	19	3	465
B05	14.6	3	2	9	6	495
B06	32.8	6	4	22	11	915
B07	11.9	5	3	14	6	435
B08	18.8	9	4	26	10	765
C00	19.6	9	4	26	10	735
C01	11.9	4	3	14	6	405
C02	32.8	9	4	22	11	945
C03	14.6	5	2	9	6	465
C04	14.7	6	1	20	3	465
C05	15.5	4	2	35	4	495
C06	23.5	4	2	102	9	705
C07	28.9	3	4	19	16	795
C08	50.4	16	10	63	27	1 395

Table 10.1: An overview of the benchmarked real-life data instances. The columns show instance identifiers, trip distance (km), number of changing speed limits, number of passage points, maximum change in altitude (m), the number of linearized track slopes, and finally the trip time (s).

					FP	LSA			Heuristic			
Case	Reg.	Vertices	Edges	Labels	Cost	Time	Sav	Arr.	Time	Sav	Arr.	Δ
A00	9	246	259	160	22.6	20	31.3%	-4.8	19	29.5%	1.6	1.8%
A01	26	1 122	1 606	1 490	31.9	118	16.8%	-6.7	10	13.7%	1.3	3.1%
A02	21	906	1 245	3 012	29.8	77	19.0%	-6.5	25	18.6%	1.8	0.4%
A03	17	664	866	1 237	22.6	68	17.5%	-6.3	13	17.5%	-1.0	-0.0%
A04	15	431	490	1 650	24.6	58	43.7%	5.1	3	39.3%	0.5	4.4%
A05	19	714	875	4 184	33.8	97	32.0%	-2.2	20	27.5%	1.0	4.5%
A07	5	350	488	405	27.1	15	34.9%	-3.8	84	35.6%	1.1	-0.7%
A08	3	357	511	409	33.6	15	37.1%	-5.2	36	44.3%	-0.2	-7.3%
A09	7	382	517	47	33.9	41	2.8%	-6.7	0	0.0%	-4.0	2.8%
A10	5	259	295	233	37.3	18	37.4%	-4.7	33	38.3%	0.3	-0.9%
A11	4	356	485	594	34.0	32	29.9%	-4.6	31	30.0%	0.0	-0.2%
A12	7	629	1 049	695	20.1	57	32.3%	-1.3	17	32.2%	-1.9	0.1%
A13	10	392	491	1 106	39.4	40	17.9%	-5.4	23	16.9%	-0.6	1.0%
A14	11	572	736	378	18.0	41	50.8%	-2.8	84	49.3%	1.5	1.5%
A16	3	43	40	54	12.6	6	83.8%	45.1	7	71.8%	-0.7	12.0%
Avg							32.5%			31.0%		1.5%
B00	61	5 756	8 900	112 756	46.2	1 248	50.6%	-4.5	23	38.9%	0.3	11.7%
B01	22	3 124	5 169	55 264	43.2	499	36.3%	-5.2	23	35.5%	0.9	0.8%
B02	14	2 156	3 310	60 223	44.0	426	46.1%	-5.3	86	41.4%	1.1	4.7%
B03	7	1 579	2 562	11 110	56.4	130	44.2%	-5.4	38	42.6%	1.8	1.6%
B04	7	1 040	1 508	4 181	52.3	49	42.8%	-4.0	27	42.4%	0.4	0.4%
B05	9	1 985	3 241	22 633	51.0	183	55.3%	-5.6	42	53.2%	-0.8	2.1%
B06	21	3 492	5 999	84 476	42.9	635	37.8%	-4.7	41	36.6%	-1.1	1.2%
B07	13	1 930	3 067	12 277	52.8	143	60.3%	-5.0	46	59.9%	-1.5	0.4%
B08	23	2 039	2 873	42 344	47.5	314	71.0%	-5.0	64	70.3%	1.7	0.7%
Avg							49.4%			46.7%		2.6%
C00	23	2 053	3 100	22 360	49.3	208	53.1%	-4.8	54	52.3%	-1.0	0.9%
C01	12	2 023	3 303	10 577	59.6	201	50.2%	-5.2	37	49.5%	-0.8	0.7%
C02	24	2 738	4 159	25 868	45.5	342	41.1%	-4.5	24	37.4%	-0.7	3.7%
C03	12	1 921	3 028	13 074	52.4	163	47.3%	-5.3	33	45.2%	1.7	2.0%
C04	7	1 088	1 608	2 854	50.3	51	44.3%	-4.6	30	43.3%	1.7	1.0%
C05	7	2 014	3 504	26 599	46.3	220	55.3%	-6.4	29	53.4%	1.3	1.9%
C06	13	1 643	2 404	6 311	44.9	160	35.0%	-5.6	22	39.5%	-1.5	-4.5%
C07	25	2 857	4 434	35 412	44.1	395	34.6%	-5.3	41	33.7%	0.9	0.9%
C08	57	5 842	9 356	93 436	38.1	1 143	35.7%	-5.7	36	35.2%	1.5	0.6%
Avg							44.1%			43.3%		0.8%

Table 10.2: An overview of the benchmark results. The first columns show the instance identifier, number of regions, vertices and edges in the underlying graph. The forth column shows the number of generated labels, and the fifth shows the energy cost for the fastest profile (*FP*). For each method we report runtime (ms), savings (relative to *FP*) and the remaining trip time (s). The final column shows the difference in savings.

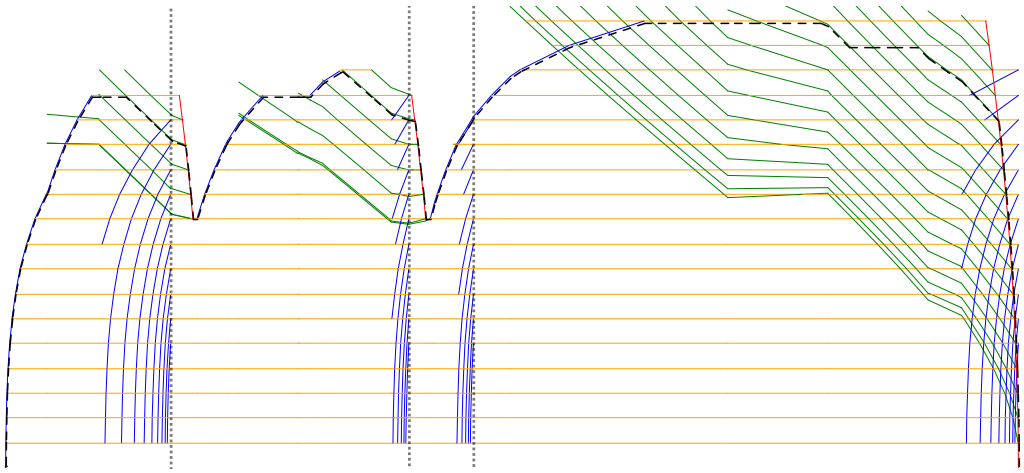


Figure 10.4: An illustration of an optimized speed profile in the graph presented in Section 10.3.2. The profile, depicted as a dashed black line, follows a series of accelerate-cruise-coast-brake stages. Multiple acceleration and braking curves are present due to speed restrictions. Many acceleration curves are inserted before passage points (vertical dotted lines) to avoid (potential) infeasibility.

these cases more carefully we conclude that the solutions are correct. The differences are a result of greedy heuristic choices when facing speed limits and passage points.

An example of a found optimized speed profile using LSA is illustrated in Figure 10.4. In the example the passage points are not very binding, however it can be observed how the solution method distributes surplus time by letting the train coast in three different segments. A second example is shown in Figure 10.5. In this example it is observed that the passage point is now more binding. An excess of surplus time is available before the first passage point, however, in order to avoid infeasibility after this point a high speed is required in the sequel.

10.6 Conclusion

Optimizing train speed profiles is important research as trains are one of the major electricity consumers in most countries. Minimizing even a small percentage leads to significant savings and reduces the environmental impact. A variety of solution methods exist in literature but further improvements can be made in terms of computational requirements and increased realism or accuracy.

We proposed a novel DP-based solution method for finding optimized speed profiles. Although it is not an exact method for finding the optimal solution, our computation results show that very high quality solutions are indeed found within 1-2 seconds in the worst case. The computational time savings are significant compared to previous methods based on DP. A large number of real-life instances have been considered. We conclude that the proposed solution method is suitable for providing real-time driving assistance.

The solution framework is elegant and flexible. The underlying graph can be extended without the use of advanced mathematics. The solver is interchangeable as the RCSPP can be solved using different methods such as Lagrangian relaxation, constraint programming or even heuristic methods.

For future research, a number of interesting extensions are worth investigating. The solution method could be extended to handle steep gradients where cruising is not possible, and extended to consider trains with discrete speed control. The potential of the presented methodology is not

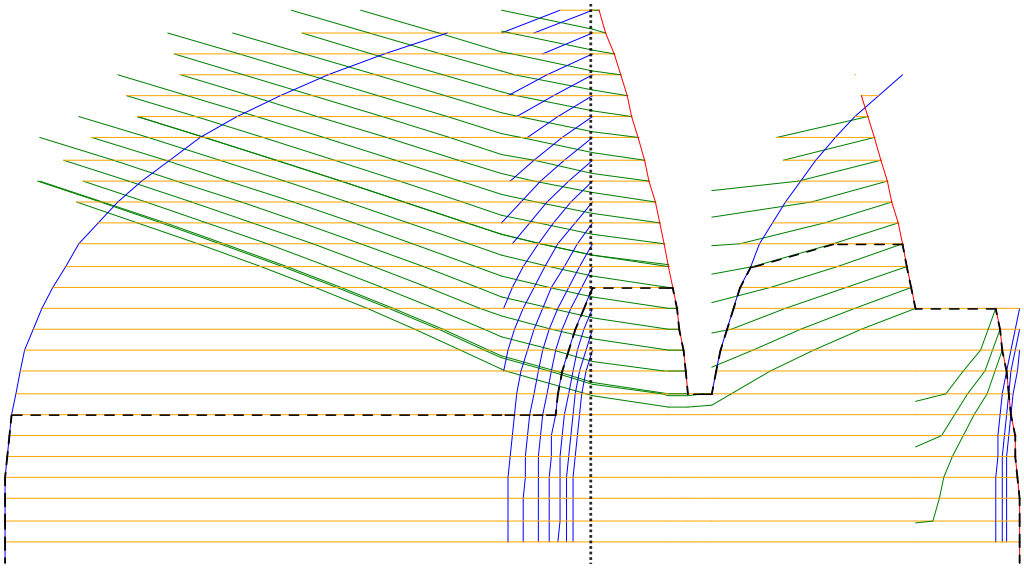


Figure 10.5: An illustration of an optimized speed profile in the graph presented in Section 10.3.2. The profile, depicted as a dashed black line, follows a series of accelerate-cruise-coast-brake stages. Multiple acceleration and braking curves are present due to speed restrictions. Many acceleration curves are inserted before passage points (vertical dotted lines) to avoid (potential) infeasibility.

fully exploited. Even more aggressive preprocessing methods could improve computational time as well as tuning of the label setting algorithm. Finally, we believe that proposed framework is eligible for other types of transportation modes, examples include trams, trucks, air-planes and vessels.

Bibliography

[1] Cubris aps. `http://http://www.cubris.dk/`. Accessed: 2015-06-25.

[2] Innovative integrated energy efficiency solutions for railway rolling stock, rail infrastructure and train operation. `http://www.railenergy.org`. Accessed: 2015-06-26.

[3] I.A. Asnis, A.V. Dmitruk, and N.P. Osmolovskii. Solution of the problem of the energetically optimal control of the motion of a train by the maximum principle. {*USSR*} *Computational Mathematics and Mathematical Physics*, 25(6):37 – 44, 1985.

[4] C.S. Chang and S.S. Sim. Optimising train movements through coast control using genetic algorithms. *Electric Power Applications, IEE Proceedings* -, 144(1):65–73, Jan 1997.

[5] Jiaxing Cheng. *Analysis of optimal driving strategies for train control problems*. University of South Australia, 1997.

[6] Andrea D'Ariano, Marco Pranzo, Ingo Hansen, et al. Conflict resolution and train speed coordination for solving real-time timetable perturbations. *Intelligent Transportation Systems, IEEE Transactions on*, 8(2):208–222, 2007.

[7] CW Davies, Ion Association, et al. Butterworths, 1962.

[8] Rudiger Franke, Peter Terwiesch, and Markus Meyer. An algorithm for the optimal control of the driving of trains. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, volume 3, pages 2123–2128. IEEE, 2000.

[9] Seong-Ho Han, Yun Sub Byen, Jong Hyen Baek, Tae Ki An, Su-Gil Lee, and Hyun Jun Park. An optimal automatic train operation (ato) control using genetic algorithms (ga). In *TENCON 99. Proceedings of the IEEE Region 10 Conference*, volume 1, pages 360–362 vol.1, 1999.

[10] Ingo Arne Hansen and Jörn Pachl, editors. *Railway Timetabling & Operations*. Eurailpress, 2008.

[11] Phil Howlett. An optimal strategy for the control of a train. *The Journal of the Australian Mathematical Society. Series B. Applied Mathematics*, 31(04):454–471, 1990.

[12] Phil Howlett. The optimal control of a train. *Annals of Operations Research*, 98(1-4):65–87, 2000.

[13] Philip G Howlett and Peter J Pudney. *Energy-efficient train control*. Springer, 1995.

[14] Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In Guy Desaulniers, Jacques Desrosiers, and MariusM. Solomon, editors, *Column Generation*, pages 33–65. Springer US, 2005.

[15] Jyh-Cherng JONG and Sloan CHANG. Algorithms for generating train speed profiles. *Journal of the Eastern Asia Society for Transportation Studies*, 6:356–371, 2005.

[16] H Ko, T Koseki, and M Miyatake. Application of dynamic programming to the optimization of the running profile of a train. In *Computers in railways IX*, 2004.

[17] Maialen Larranaga, Jonatha Anselmi, Urtzi Ayesta, Peter Jacko, and Asier Romo. Optimization techniques applied to railway systems. Technical report, Basque Center for Applied Mathematics, January 2013. 17p.

[18] Shaofeng Lu. *Optimising power management strategies for railway traction systems.* PhD thesis, University of Birmingham, 2011.

[19] Horst Strobel, Peter Horn, and Manfred Kosemund. Contribution to optimum computer-aided control of train operation. In *Proc. 2nd IFAC/IFIP/IFORS Symp. Traffic Control and Transportation Systems, Monte-Carlo*, pages 377–387, 1974.

[20] Yihui Wang, Bing Ning, Fang Cao, Bart De Schutter, and Ton JJ Van den Boom. A survey on optimal trajectory planning for train operations. In *Service Operations, Logistics, and Informatics (SOLI), 2011 IEEE International Conference on*, pages 589–594. IEEE, 2011.

[21] Liu Wei, Li Qunzhan, and Tang Bing. Energy saving train control for urban railway train with multi-population genetic algorithm. In *Information Technology and Applications, 2009. IFITA'09. International Forum on*, volume 2, pages 58–62. IEEE, 2009.

[22] KK Wong and TK Ho. Coast control for mass rapid transit railways with searching methods. *IEE Proceedings-Electric Power Applications*, 151(3):365–376, 2004.

Chapter 11

Energy-Efficient Passenger Train Departure Synchronization

Jørgen Thorlund Haahr* Martin Philip Kidd*

*Department of Management Engineering, Technical University of Denmark,
Produktionstorvet, Building 426, DK-2800 Kgs. Lyngby, Denmark
jhaa@dtu.dk, dapi@dtu.dk

[1] **Abstract** Timely recuperation of energy through regenerative braking is crucial in order to ensure energy efficient railway timetables. This requires a careful synchronisation of train departures such that high energy peaks, as a result of simultaneously accelerating trains, can be avoided. In this report we consider a variant of this problem as presented in the FAU Open Research Challenge in Discrete Optimization. We propose a mixed integer linear programming formulation (MILP) together with a number of heuristics based on this model. We show that the MILP can obtain optimal solutions to most of the instances proposed as part of the challenge, and that the matheuristics can find good solutions in short computation times.

11.1 Introduction

Rail is one of the major consumers of energy, and there is a desire for railway operators to be energy-efficient in their daily operations. Towards this aim a significant component of a train operating company's electricity bill usually depends on the highest peak of energy usage within a given period, a measure imposed to encourage operators to maintain a balanced distribution of power consumption throughout their operations [2].

Compared to the case where, for example, trains are billed individually based on their power consumption, this measurement results in more complex timetabling problems since the energy peaks are highly dependent on the interactions between trains. Synchronization becomes important in order to ensure that energy regenerated (when trains are braking) is efficiently recuperated by accelerating trains.

In this report we consider the Energy-Efficient Train Timetables Problem (EETTP), which comprises the construction of a timetable such that the period with the highest peak power usage is minimized. The problem was presented and instances were provided by the Chair of Economics,

[1] Technical Report *(July 2015)*

Discrete Optimization and Mathematics at Friedrich-Alexander-University Erlangen-Nürnberg (FAU), in the Discrete Optimization part of their *Open Research Challenge*[2].

In this paper we present a mixed-integer programming formulation (MILP) together with a number of heuristic approaches based on this formulation. Our approach is based on *dwell time control*, i.e., deciding only the departure times of trains while assuming fixed running times. Albrecht [1], for example, considers the same problem, but uses an approach for train running time control instead of train dwell time control. Sansó & Girard [6] considers the problem with dwell time control within the context of the Montreal metro system, and present a formulation in some ways similar to the one presented in this paper. However, due to the fact that we consider a more complex network structure, we model safety (headway) constraints differently and also include constraints for maintaining passenger connections which they do not. A general overview of energy efficiency in railways is given by Albrecht [2]. For a general overview of railway timetabling see Cacchiani & Toth [4].

We note that the problem under consideration in this report is similar to the Resource Leveling Problem (RLP) in project scheduling [5]. Arrivals and departures of trains may be viewed as project activities, while headway, runtime, dwell time and connection constraints may be viewed as precedence relations between activities. Moreover, electricity may be viewed as a single resource, the usage of which needs to be flattened out (leveled) over the scheduling horizon. The RLP and EETTP differ mostly with respect to the objective function, since resource leveling problems usually focus on specific objective functions that differ from the one considered in this paper. The RLP has been well studied in the literature, and an overview of exact methods for both continuous and discrete time variants is given by Rieck & Zimmermann [5].

The presented problem has a nice structure that can easily be modeled using a linear program with binary variables. We therefore investigate the potential of solving such a model with existing state-of-the-art solvers. The problem is fairly constrained, and our intuition is that variable/constraint preprocessing and constraint propagation may be key in solving the problem. Furthermore, an exact approach may provide bounds on optimality that can be used to evaluate the effectiveness of heuristic approaches. Finally, we also investigate matheuristic approaches as they are able to combine strengths of both worlds (exact and heuristic approaches).

This paper is organized as follows. In Section 11.2 we give an informal problem description, followed by a more formal description in terms of a MILP in Section 11.3. We propose a number of matheuristics in Section 11.4 and we compare all methods with respect to the provided instances in Section 11.5. We conclude the report with a summary in Section 11.6 and ideas for future research in Section 11.7.

11.2 Problem Description

In input we are given a set of trains and a railway network. For each train we are given a route in the network, which is a sequence of legs connecting a sequence of stops. For each leg a departure window is given together with a power profile for traversing the leg. Here, a power profile defines the power phases of a train on a trip, i.e., when a train is consuming or regenerating power during the trip. The problem then amounts to deciding the times of the departures of all legs such that the highest peak of electrical power usage is minimised.

The choice of departure times for the trains is governed by minimum dwell-time constraints at platforms and headway time constraints between trains traveling on the same track in the same direction. Further constraints include a fixed order in which trains are to traverse each track in the network, and the fact that passenger connections between certain trains at certain stations need to be maintained. Here, a passenger connection is defined whenever the difference in time

[2]See `https://openresearchchallenge.org/discreteOptimization/ChairofEconomics/The+Challenge`.

between an arrival and departure at the same station falls within a 5–15 minute interval in the original timetable. We consider a discretization of time such that trains can only depart on whole minutes, and we assume fixed running times and single-direction tracks.

Given a solution which specifies the departure times for all legs, the power profiles are used to calculate the net power usage of all trains for each *second* of the considered time horizon. Within a power profile each second can be either positive (acceleration) or negative (regeneration). For consecutive periods of 15 minutes, covering the entire considered time horizon, the average power consumption is calculated for each period by only taking into account seconds with a positive net power usage (the assumption is thus made that power regenerated by a braking train that is not consumed by an accelerating train in the span of the same second is lost). The objective is to minimize the 15-minute period with the highest average power consumption.

11.3 MILP Formulation

We are given a set of legs $\mathbb{L}$ together with a list of consecutive possible departure times $\mathbb{T}_i \subseteq \mathbb{T}$ for each leg $i \in \mathbb{L}$, where $\mathbb{T}$ is the set of discrete time instants in the considered time horizon. We consider the problem of deciding at which time instant each leg should depart so as to minimize peaks in energy consumption, subject to constraints ensuring that runtimes, dwell times, headways and connections are respected. To this aim we define a binary decision variable x_t^i which assumes a value of 1 if leg $i \in \mathbb{L}$ departs at time $t \in \mathbb{T}_i$ and 0 otherwise. First we impose constraints to select exactly one of the possible departures:

$$\sum_{t \in \mathbb{T}_i} x_t^i = 1 \qquad i \in \mathbb{L}. \tag{11.1}$$

The track headway, passenger connections, dwell time, and runtime constrains share a common *precedence* structure, which is why we model time using the same notation. In order to model the constraints on departure times we consider a directed graph $G = (\mathbb{L}, \mathbb{A})$ where $(i, j) \in \mathbb{A} \subseteq \mathbb{L} \times \mathbb{L}$ denotes a *precedence constraint* on the departure time of j with respect to the departure time of i. Each precedence constraint $(i, j) \in \mathbb{A}$ is associated with a set of *conflicting* pairs of time instants $\mathbb{C}_{ij} \subseteq \mathbb{T}_i \times \mathbb{T}_j$, where $(t, t') \in \mathbb{C}_{ij}$ indicates that a conflict will arise if leg i departs at t and leg j departs at time t'. The precedence constraints may then be modelled by one of the following two inequality sets:

$$\sum_{t' \in \mathbb{T}_j : (t,t') \in \mathbb{C}_{ij}} x_{t'}^j + x_t^i \leq 1 \qquad (i, j) \in \mathbb{A}, t \in \mathbb{T}_i. \tag{11.2}$$

$$\sum_{t' \in \mathbb{T}_i : (t,t') \in \mathbb{C}_{ij}} x_{t'}^i + x_t^j \leq 1 \qquad (i, j) \in \mathbb{A}, t \in \mathbb{T}_j. \tag{11.3}$$

Note the subtle difference between the two inequality sets, and the fact that using only one set is sufficient. Including both sets can strengthen the LP relaxation, but in our experiments we found this to be of no significant benefit.

In the subsection that follows we discuss in more detail the construction of the sets $\mathbb{A}$ and $\mathbb{C}_{ij}$, and in the section after that we discuss the objective function.

11.3.1 Precedence Constraints

First of all, the fixed runtime and dwell time (at the destination) of each leg should be respected. Let r_i denote the running time and d_i the dwell time of leg $i \in \mathbb{L}$. In input we have for each

leg a *train successor* leg which represents the next movement of the same train. If j is the train successor leg of i, then $(i, j) \in \mathbb{A}$ and

$$\mathbb{C}_{ij} = \{(t, t') \in \mathbb{T}_i \times \mathbb{T}_j \mid t + r_i + d_i > t'\}.$$

Secondly, headways need to be respected between trains entering the same track. Note, we assume that no track is used in both directions. In input we have for each leg a *track successor* leg that will follow it on the same track (recall that the order in which trains traverse the tracks cannot be altered). If j is the track successor leg of i, then $(i, j) \in \mathbb{A}$ and

$$\mathbb{C}_{ij} = \{(t, t') \in \mathbb{T}_i \times \mathbb{T}_j \mid t + h_{ij} > t'\},$$

where h_{ij} denotes the required headway between the consecutive legs i and j.

Finally, connections need to be maintained. In input we have for each leg a *connection successor* leg, that has to maintain a connection with it within a specified time window. If j is the connection successor leg of i, then $(i, j) \in \mathbb{A}$ and

$$\mathbb{C}_{ij} = \{(t, t') \in \mathbb{T}_i \times \mathbb{T}_j \mid t + r_i + c_{ij}^{\min} > t' \ \vee \ t + r_i + c_{ij}^{\max} < t'\},$$

where $c_{ij}^{\min}$ and $c_{ij}^{\max}$ denote the minimum and maximum time j should wait after the arrival of i.

11.3.2 Objective Function

The set $\mathbb{T}$ will be a discretization of time in minutes, but for the calculation of the objective function we consider a finer discretization of time in seconds, represented by the set $\mathbb{S}$.

Let $\pi_{st}^i \in \mathbb{R}$ denote the power usage[3] during second $s \in \mathbb{S}$ by leg $i \in \mathbb{L}$, given that it departs at time $t \in \mathbb{T}_i$ (calculated from its power profile given in input). Let $\pi_s \in \mathbb{R}_0^+$ be a decision variable representing the total power consumption during second $s \in \mathbb{S}$, realized by the constraint set

$$\sum_{i \in \mathbb{L}} \sum_{t \in \mathbb{T}_i} \pi_{st}^i x_t^i \leq \pi_s \qquad s \in \mathbb{S}. \tag{11.4}$$

Note that if the left hand side of the above inequality is negative, this represents lost energy. Consequently, $\pi_s = 0$ in this case, as π_s is defined as a non-negative continuous variable.

As stated before, the objective is to minimize the 15 minute time window with the highest average power consumption. Towards this end, let $\mathbb{P}$ be the set of consecutive 901 second periods that covers the considered time window, where any two consecutive periods overlap in exactly one second. Let f_p and ℓ_p denote the first and last second in period $p \in \mathbb{P}$, respectively, and let Π be the total energy consumption of the period with the highest consumption. This value is realized by the constraint set

$$\frac{1}{2} \sum_{s=f_p}^{\ell_p - 1} (\pi_s + \pi_{s+1}) \leq \Pi \qquad p \in \mathbb{P}, \tag{11.5}$$

using the trapezoidal rule for approximating integrals.

Finally, the MILP formulation is given by

$$\begin{gathered} \text{minimize } \Pi \\ \text{subject to (11.1)–(11.5).} \end{gathered}$$

[3]By *power usage* we mean either consumption ($\pi_{st}^i > 0$), generation ($\pi_{st}^i < 0$), or that the leg simply does not cover second s if it departs at time t ($\pi_{st}^i = 0$).

11.4 Heuristic Approaches

In this section we describe two heuristic approaches to the problem, both making use of variants of the MILP presented in the previous section in different ways. A third heuristic will also be used, namely imposing a time limit on the solution of the MILP presented in the previous section.

11.4.1 Heuristic Approximation of Π

The calculation of the objective function requires a large amount of variables to be included in the MILP model, namely a variable π_s for each second $s \in \mathbb{S}$ in the considered time horizon. This is necessary to ensure that a negative net power usage during any second results in energy lost to the system.

We consider an aggregation of these variables, thereby obtaining a relaxation of the problem by assuming that regenerated energy is not lost and can be recuperated at any other point in time. This is, of course, not a realistic assumption, but it results in a MILP with less variables and constraints that can be solved in considerably less time. Full accuracy is lost because the power at every second is no longer restricted to be non-negative.

In order to achieve this, we define, for each leg $i \in \mathbb{L}$, time instant $t \in \mathbb{T}$ and period $p \in \mathbb{P}$, the quantity

$$\pi^i_{pt} = \frac{1}{2} \sum_{s=f_p}^{\ell_p - 1} \left(\pi^i_{st} + \pi^i_{s+1\,t} \right)$$

i.e., the total power contribution of leg i in period p, given that i departs at time t. Note, that π^i_{st} and π^i_{pt} are not variables but coefficients in the model.

Now constraint sets (11.4) and (11.5) may be removed and replaced by

$$\sum_{i \in \mathbb{L}} \sum_{t \in \mathbb{T}_i} x^i_t \pi^i_{pt} \leq \Pi \qquad p \in \mathbb{P}. \tag{11.6}$$

Note that the solution given by the MILP in this case will both give a lower bound to the optimal solution of the original MILP, as well as a heuristic solution to the problem.

11.4.2 Rolling Horizon Matheuristic

In this section we propose an iterative matheuristic, which, during each iteration, attempts to improve the incumbent solution by focusing on the period with the highest average power consumption in the incumbent, henceforth referred to as the *focus period*. The matheuristic considers a restricted time window that includes the focus period together with a certain number of periods around it (governed by an input parameter), and solves the EETTP inside this window using a local search heuristic. The local search heuristic itself uses the MILP to make small changes to the incumbent solution as long it improves the objective function value. This MILP is a restricted version of the presented MILP in Section 11.3, where a selected set of departures (around the focus period) are only allowed to be slightly shifted in time with respect to the incumbent (i.e., the departure windows are trimmed).

In order to impose a rolling horizon, we restrict the considered time horizon to some window $[T_{\min}, T_{\max}] \subseteq \mathbb{T}$, and only allow legs with departure windows completely inside this window to be altered, while all other departures stay fixed. The set of fixed legs are then given by

$$\hat{\mathbb{L}} = \{i \in \mathbb{L} : \mathbb{T}_i \cap [T_{\min}, T_{\max}] \subset \mathbb{T}_i\},$$

and we impose the additional constraints

$$x_t^i = \hat{x}_t^i, \qquad i \in \hat{\mathbb{L}}, \tag{11.7}$$

given a solution $\hat{\boldsymbol{x}} = [\hat{x}_t^i]$ to (11.1)–(11.2). In other words, all departure decisions outside the considered horizon are fixed to match the incumbent.

In order to trim the departure windows according to $\hat{\boldsymbol{x}}$ for the purpose of applying the local search heuristic, we impose the constraints

$$\sum_{t \in \mathbb{T}_i(\hat{\boldsymbol{x}})} x_t^i = 1, \qquad i \in \mathbb{L}, \tag{11.8}$$

where

$$\mathbb{T}_i(\hat{\boldsymbol{x}}) = \left\{ t \in \mathbb{T}_i : \left| \sum_{t' \in \mathbb{T}_i} t' \hat{x}_{t'}^i - t \right| \leq \tau \right\},$$

for some parameter τ. In other words, departure times in the new solution can differ in at most τ minutes from the corresponding times in the incumbent.

The matheuristic is formally described in Algorithm 6. The algorithm starts from an initial solution[4] $\hat{\boldsymbol{x}}$ and initial restricted time window of 1, which indicates the number of periods on either side of the focus period to include in the rolling horizon. The next rolling horizon window is determined in Steps 5–7 and the local search is performed in Steps 8–11. During each iteration the local search solves the MILP including the constraints for the rolling horizon as well as for trimming the departure windows. It continues as long as it can find improved solutions for the current horizon. The stretch stays fixed as long as the local search can find improved solutions, after which the stretch is increased and the process repeated. The entire process is repeated until the stretch reaches a maximum value, and then the matheuristic terminates.

In what follows we will use the notation $T\tau A\delta$ to denote the specific configuration of the rolling horizon matheuristic where the parameter τ has the same meaning as above and where the horizon $[T_{\min}, T_{\max}]$ covers δ periods completely, with the focus period in the middle.

Algorithm 6 Rolling horizon matheuristic

Input: Instance of the EETTP, trim size τ, max window stretch $n_{\max}$
Output: A solution $\hat{\boldsymbol{x}}$ to the instance
1: Generate initial solution $\hat{\boldsymbol{x}}$
2: $n \leftarrow 1$
3: **while** $n \leq n_{\max}$ **do**
4: **while** improvement found **do**
5: $p^* \leftarrow$ period with the highest average power consumption
6: $T_{\min} \leftarrow T_{p^*} - 900n$
7: $T_{\max} \leftarrow T_{p^*} + 900n$
8: **while** improvement found **do**
9: $\tilde{\boldsymbol{x}} \leftarrow$ minimize Π subject to (11.1)–(11.8)
10: **if** $\Pi_{\tilde{\boldsymbol{x}}} < \Pi_{\hat{\boldsymbol{x}}}$ **then**
11: $\hat{\boldsymbol{x}} \leftarrow \tilde{\boldsymbol{x}}$
12: $n \leftarrow n + 1$

[4]The provided instances all came with initial feasible solutions, and we used these solutions as starting points for the matheuristic.

11.5 Results

The complete MILP model and a set of matheuristic have been tested using the provided instances, and in this section we report the results. To our surprise the complete MILP model appears to be quite solvable, and produces superior results. The heuristic approaches can be faster but never better. We show the resulting solution quality that can be achieved using heuristics. Note that the MILP used in the implementation of the matheuristics is programmed to be as efficient as possible; it does not contain variables with fixed values nor constraints without variables.

The best results (which also are the submitted results) were produced by solving the complete MILP model using CPLEX 12.6. An overview of the statistics are shown in Table 11.1. In all cases, except the first, we were able to obtain solution with a proven gap less than 0.5%. In 8 of the 11 cases we have solutions that have a proven optimality gap of 0.0001%, thereafter the search terminates. All produced results are obtained within a maximal runtime of 24 hours. Instance 7 and 10 run out of memory (24GB) before 24 hours are reached, and the solutions for both instances are therefore obtained in less time. In contrast to our expectations, the first instance proves to be hard to solve. We believe that the strength of the MILP model is largely due to preprocessing and constraint propagation; experimental runs with finding the LP relaxation turns out to be more time-consuming than running the MILP formulation for some instances (Table 3)

Instance	Cost	Gap	Runtime	Variables	Constraints	Nodes
1	1.09	15.61%	86 410	16 737	16 763	117 817
2	2.94	0.00%	2 681	16 981	16 959	1 833
3	15.18	0.00%	435	17 748	17 835	6 271
4	14.97	0.00%	1 361	20 008	20 795	23 421
5	19.42	0.00%	1 499	21 306	21 786	12 870
6	20.43	0.00%	5 687	20 265	20 574	138 556
7	16.32	0.11%	28 825	25 927	29 366	62 280
8	2.42	0.00%	7 092	41 163	114 433	8 996
9	98.14	0.00%	18 003	27 907	30 413	61 451
10	68.12	0.01%	43 220	33 702	38 849	127 267
	259.03					

Table 11.1: Overview of best obtained results

Granted, it is hard to improve upon the result produce by solving the complete MILP model. However, shorter runtimes are in some cases important. Furthermore, in face of even larger instances, the MILP may become impractical. Four different heuristic variants are chosen, the results are shown in Table 11.2. The first two methods are *T1A5* and *T3A9* (using the notation introduced in Section 11.4.2). The third heuristic is the one presented in Section 11.4.1 that uses a heuristic approximation of the objective function, while the forth method is the full MILP together with a time limit of 15 minutes. We observe that the last two methods produce the best overall results, and the amount of time to solve the problem is relatively low. The results of the other approaches show that the methods are able to produce solutions within short time using the heuristic described. To our surprise, the matheuristics presented in Section 11.4 did not prove to be more effective than the MILP in section 11.3, even with if used with a heuristic objective.

11.6 Conclusion

We have presented several solution methods for solving the EETTP. One exact and several heuristic approaches have been proposed and benchmarked.

	T1A5		T3A9		Heuristic Objective		MILP 15min	
Instance	Cost	Runtime	Cost	Runtime	Cost	Runtime	Cost	Runtime
1	1.20	53	1.17	876	1.46	6	1.17	901
2	3.44	49	3.17	169	3.77	1	2.99	901
3	19.50	29	17.92	131	15.29	2	15.18	366
4	19.00	30	17.51	397	15.33	3	14.97	901
5	25.84	65	23.82	405	20.06	4	19.42	901
6	25.81	43	22.92	673	20.92	49	20.45	902
7	20.53	109	16.47	5 781	18.23	492	16.86	902
8	2.68	18	2.67	62	2.71	29	2.61	902
9	107.98	188	101.81	5 298	98.99	63	98.26	904
10	79.25	17	68.61	11 000	68.14	581	68.72	903
	305.23	601.00	276.08	24 791	264.90	1 230	260.64	8 484

Table 11.2: Overview of four different heuristic solution methods.

The exact method, based on a MILP model, performs very well. The best found solutions are obtained with this method. Even if a short time limit is given, e.g., 15 minutes per instance, the exact method provides superior solutions. Using a heuristic objective good results can be obtained faster. The matheuristic approaches show a general lack of strength. The solution quality is poor compared to the MILP model with a 15 minute time limit, and given more flexibility the runtime becomes too high.

To our surprise, the exact MILP model approach is found to be superior to the other considered approaches. In addition to the high quality solution obtained, this exact method is also able to provide optimality bounds. The quality of any solutions found by heuristic approaches will always be unproven without good bounds.

11.7 Future Research

As mentioned in the introduction, the precedence constraints considered in this report is similar to precedence constraints in project scheduling problems, and can be modeled using similar techniques. Artigues *et al.* [3] present a large variety of MILP models for the *resource constrained project scheduling problem*, where they refer to the time-indexed variables that we have used in our model as "pulse" start variables. They presented a number of different time-indexed variables — such as whether or not a specific activity starts before a specific point in time (start "step" variables), or whether or not a specific activity is in progress at a specific point in time (on/off variables) — and these ideas could lead to a number of alternative formulations for the EETTP. As future research we propose further investigations into alternative MILP approaches inspired by these ideas.

Instance	Objective	Runtime	Variables	Constraints
1	735.34	143	16 737	16 844
2	2 640.36	30	16 981	17 069
3	13 600.85	195	17 748	18 110
4	13 411.34	228	20 008	21 830
5	17 350.00	259	21 306	22 640
6	18 317.91	141	20 265	21 284
7	14 607.51	2 013	25 927	33 507
8	2 032.33	97 767	41 163	188 728
9	88 100.69	2 163	27 907	34 276
10	61 044.77	2 934	33 702	45 560

Table 3: Results of solving the LP relaxations, using both (11.2) and (11.3) in the MILP.

Bibliography

[1] T Albrecht. Reducing power peaks and energy consumption in rail transit systems by simultaneous train running time control. In Eduardo Pilo, editor, *Power Supply, Energy Management and Catenary Problems*, pages 3–12. WIT Press, Southampton, 2010.

[2] T Albrecht. Energy-efficient train operation. In I A Hansen and J Pachl, editors, *Railway Timetabling & Operations*, pages 91–116. Eurailpress, Hamburg, 2nd edition, 2014.

[3] C Artigues, O Koné, P Lopez, and M Mongeau. Mixed-integer linear programming formulations. In C Schwindt and J Zimmermann, editors, *Handbook on Project Management and Scheduling, Volume 1*, pages 17–41. Springer International Publishing, Cham, 2015.

[4] Valentina Cacchiani and Paolo Toth. Nominal and robust train timetabling problems. *European Journal of Operational Research*, 219(3):727–737, 2012. Feature Clusters.

[5] J Rieck and J Zimmermann. Exact methods for resource leveling problems. In C Schwindt and J Zimmermann, editors, *Handbook on Project Management and Scheduling, Volume 1*, pages 361–387. Springer International Publishing, Cham, 2015.

[6] Brunilde Sansó and Pierre Girard. Instantaneous power peak reduction and train scheduling desynchronization in subway systems. *Transportation Science*, 31(4):312–323, 1997.

Appendix A

The ROADEF Challenge 2014 Solution Framework

Jørgen Thorlund Haahr* Simon Henry Bull*

*Department of Management Engineering, Technical University of Denmark,
Produktionstorvet, Building 426, DK-2800 Kgs. Lyngby, Denmark
jhaa@dtu.dk, simbu@dtu.dk

[1] **Abstract** The following document was submitted, along with a solution framework. The document briefly describe the framework used to produce solution the obtained solution for the challenge.

A.1 Overview

The proposed solution framework can be classified as a *math heuristic* as it combines exact with heuristic methods. The heart of the solution framework is an Simulated Annealing (SA) framework that iteratively destroys and builds random train routes. However, in order to improve convergence (and runtime) a few smaller Mixed Integer Program (MIP) problems are solved in advance in order to avoid resource use conflicts and improve resource utilization. Figure A.1 illustrates the flow (and main components) of the solution framework. The full Rolling Stock Unit Management on Railway Sites (RSUM) problem is decomposed into four sequential steps which will be described in the following sections of this document. In the first step arrivals and departures are matched in order to get the best possible matching, such that e.g. the number of cancellations is minimized. Next, a platform slot is reserved for all arrivals and departures such that as many as possible are assigned to preferred platforms. Thirdly, a track group usage pattern is chosen for all arrival/departure sequences, such that no pairs of patterns are in conflict and such that no pattern is in conflict with the pre-specified imposed resource usages. Fourthly, non-overlapping facility usage slots are reserved for all maintenance activities (these are generated as a results of the matching). Finally, an SA approach iteratively removes and reroutes a group of *related* (see section A.4) trains as specified by the found matching.

[1] Technical Report *(July 2015)*

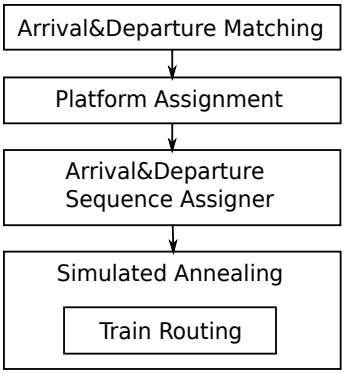


Figure A.1: An overview of the solution framework flow

A.2 Matcher

The first subproblem in the solution framework tries to match departures with compatible trains, i.e., initial trains or arrivals. The matching is formulated as a mathematical model consisting of linear constraints (and an objective) and solved using column generation, due to the large number of variables. The primary goal is to minimize the number of uncovered departures while a secondary goal is to maximize train re-uses.

The objective is formulated as a minimization of the number of unmatched departures and cost of non-satisfied train reuse :

$$\begin{aligned}\min & \sum_{d \in D} cancellationCost_d \cdot c_d \\ & + \sum_{d \in D} \sum_{t \in Comp(d)} reuseCost_t^d \cdot m_t^d \\ & + \sum_{p \in P} reuseCost_p \cdot \lambda_p\end{aligned}$$

A few heuristic artificial heuristic costs are also be added to improve the ability to perform the routing afterwards. The constraints are:

$$\sum_{t \in Comp(d)} m_t^d + \sum_{p \in P} \alpha_p^d \lambda_p \geq 1 - c_d \qquad \forall d \in D \tag{A.1}$$

$$\sum_{t \in Comp(d)} m_t^d + \sum_{p \in P} \alpha_p^d \lambda_p + \sum_{t \in Comp(d)} Block_d^t \cdot m_t^d + \sum_{p \in P} Block_p^d \cdot \lambda_p \leq 1 \qquad \forall d \in D \tag{A.2}$$

$$\sum_{d \in Comp(t)} m_t^d + \sum_{p \in P} \beta_p^t \lambda_p \leq 1 \qquad \forall t \in T \tag{A.3}$$

$$\sum_{d \in D} \sum_{t \in Comp(d)} Maint_{t,d}^{day} \cdot m_t^d + \sum_{p \in P} Maint_p^{day} \lambda_p \leq MaintLimit_{day} \quad \forall day \in H \tag{A.4}$$

Where m_t^d is a binary variable indicating whether train t is matched to departure d. The binary variables c_d indicate whether departure d is canceled or not. The binary variables λ_p indicate

whether a linked-departure pattern p is chosen. Since it is not trivial to model linked departures the m_t^d variables only indicate choices for non-linked matches while all linked arrival and departures are modeled using patterns. A pattern is the arrival/departure trajectory of a *real* train, i.e., a pattern is a sequence of matches where all (except possibly the last) have non-linked departures. A full enumeration of these patterns is intractable which is why column generation is used, the column generation process is describe in subsection A.2.1.

The cancellation cost also includes the cost of a non-satisfied reuse, if such is present. The sets D, T and H respectively represent all departures, all trains and all days of the planning horizon. The $Maint_{t,d}^{day}$ and $Maint_p^{day}$ coefficients denote how many maintenance operations are needed at day $day \in H$. The $Block_d^t$ and $Block_p^d$ indicate whether the assignment blocks departure d. The coefficients α_p^d and β_p^t respectively denote whether a pattern contains departure d or train t. The set $Comp(d)$ is the set of trains which are compatible with d, likewise the $Comp(t)$ is the set of departures that are compatible with train t. These two sets are generated in a preprocessing step. Constraints (A.1) ensure that every departure is assigned to some train, unless there is a cancellation. Constraints (A.2) ensure that at most one train is assigned to every departure, and also block departures for trains that assume that the departure is cancelled. Note that where t is a linked train m_t^d assumes that the linked departure d' is cancelled. Constraints (A.3) ensure that each train is assigned at most once. Finally, Constraints(A.4) ensure that the total number of maintenance operations (every day) is respected. For simplicity it is here assumed that, given a matching (t,d), the day that a maintenance operation is performed is fixed. With the additions of more variables, it is possible to make the day of maintenance operations a choice.

Instead of enumerating all possible train and departure matches the sets $Comp(d)$ and $Comp(t)$ are computed. First, any pairs with a train t arriving after the departure d are removed as such clearly cannot be matched. Second, all pairs where the train is not compatible with the departure are removed. Third, pairs are removed by inspecting the timespan between train and departure and comparing this to the minimum time required for routing and required maintenance appointments. Finally, all arrivals (or departures) are removed where there exists no feasible arrival (or departure) sequence, due to imposed resources. Since runtime is scarce a number of heuristic choices can also be employed here. In order to reduce the MIP size the solution framework furthermore removes matches longer than a certain timespan. Due to the team size and time all joint arrivals and departures are removed.

A model (solved in a full branch-and-price framework) can be used to generate a true lower bound on uncovered departures, or even a lower bound on the minimum cost. Such a lower bound could prove useful for heuristic methods.

A.2.1 Column Generation Subproblem

Column generation is a well-described technique used with success for solving MIP problems, e.g. the Vehicle Routing Problem with Time Windows (VRPTW). It is assumed that the reader is familiar with column generation solution methods. The subproblem can be solved as a Resource Constrained Shortest Path Problem (RSCPP). The underlying graph consists of one node per linked arrival and departure pair in addition to one source and one sink node, see Figure A.2. The edges constitute matching choices. Three families of edges are added. First, edges originating from the source to every node in the graph represent compatible arrivals that are matched to the linked departure of the node. Second, edges are added between nodes that represent compatible linked continuations, i.e., linked arrival/departures that connect to another linked arrival/departure. An example: In the (π, A, D, ω) path the departure of the initial train matching (represented by edge (π, A)) is linked to one arrival (node A). The departure of the next matching ((A, D)) is to another linked arrival (node D). The departure of the last matching is not linked to any arrival, and thus the sequence ends.

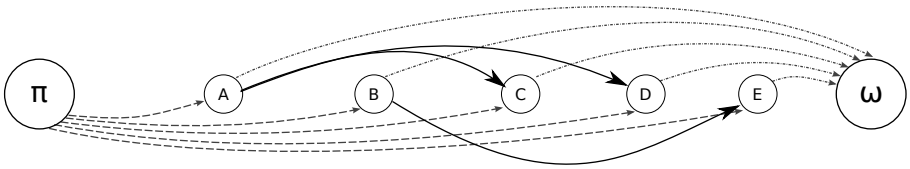


Figure A.2: An illustration of the underlying graph for the matching subproblem

The problem is a RSCPP due to the maintenance constraints. In addition to the objective coefficients, the duals from Constraints (A.1) and (A.2) are added to edges of the corresponding departure, and the duals from Constraint (A.3) are added to edges of the corresponding trains. The appropriate dual from Constraints (A.4) is added every time a maintenance operation is scheduled. Labels in the Shortest Path enumeration method keep track of total cost, remaining DBM and remaining TBM. Domination is possible if a label has lower cost ($\leq$), and at least equal remaining DBM and TBM ($\geq$). Labels originate from the source vertex and are extended by traversing available arcs and deciding whether to perform maintenance (DBM or TBM or both).

A.3 Platform Assigner

Platforms must be assigned to all covered arrivals and departures. Once an arrival/departure matching is known, a platform assignment can be performed. In a highly utilized network the arrivals and departures may be competing for the same platforms, which motivates an exact solution approach. In the solution framework a MIP is formulated that assigns one compatible platform to every covered arrival and departure:

$$\max -\sum_{a \in A} cancellationCost \cdot s_a - \sum_{d \in D} cancellationCost \cdot s_d + \sum_{(i,j) \in NC} c \cdot d_{i,j}$$

The constraints are:

$$\sum_{p \in Comp(a)} x_a^p \geq 1 - s_a \qquad \forall a \in A \tag{A.5}$$

$$\sum_{p \in Comp(d)} x_d^p \geq 1 - s_d \qquad \forall d \in D \tag{A.6}$$

$$x_i^p + x_j^p \leq 1 \qquad \forall p \in P, (i,j) \in C \tag{A.7}$$

$$end_i + M(1 - x_i^p) + d_{i,j} \leq begin_j + M(1 - x_j^p) \qquad \forall p \in P, (i,j) \in NC \tag{A.8}$$

$$\tag{A.9}$$

Two sets of binary variables are used x_a^p and x_d^p respectively indicating whether arrival a or departure d is assigned to platform p. A set of continuous variables $d_{i,j}$ exists for measuring the (approximate) slack between two consecutive platform usages. The ideal objective is to maximize the smallest slack variable. However, for performance reasons the objective is changed to maximize the slack sum (i.e. average). A coefficient $c \in \mathbf{R}$ is used for scaling the distance. C is the set of all usage pairs that are overlapping in time. NC denotes all pairs of consecutive (in time) usages. Constraints (A.5)-(A.6) ensure one assignment (or cancellation). Constraints (A.7) ensures that two assignments (arrival or departure) cannot be assigned to the same platform if they overlap in time. Constraints (A.8) measure the slack between two consecutive events if they appear on the same platform. The $start_j \in \mathbf{R}$ and $end_j \in \mathbf{R}$ are determined by the minimal usage required for the corresponding arrival or departure usage.

Note that an arrival or a departure is not limited to using the assigned platform, however the found platform assignment will ensure a minimal cancellation due to lack of a available platforms.

A.4 Simulated Annealing

In the final step a SA approach is used to search for a good solution. The initial solution is an empty solution and in every iteration a train is selected for routing, and a randomized path is generated for the train (based on the matching and allocated resource allocations). Before routing the neighborhood of the selected train is also removed (if the path is blocked). The neighborhood is the set of other trains in the current solution that intersect usages on the selected path. After adding the generated path for the selected train the neighborhood is re-inserted (if possible) in random order. The new solution is then accepted or rejected depending on new solution cost and the current temperature. An overview is shown in Algorithm 7.

Algorithm 7 Simulated Annealing

```
1: current ← GenerateEmptySolution()
2: T ← T_init
3: while T_term < T do
4:     for i ∈ {0, ..., iterations} do
5:         t ← RandomTrain()
6:         p ← RandomPath(t)
7:         n ← FindNeighborhood(p)
8:         solution' ← current
9:         solution' ← Destroy(solution', p)
10:        solution' ← Destroy(solutoin', n)
11:        solution' ← Route(solution', p)
12:        solution' ← Route(solution', n)
13:        δ ← Cost(solution') − Cost(current)
14:        if δ < 0 ∨ Random(0, 1) < e^(−δ/T) then
15:            current ← solution'
16:    T ← T · α
```

A.4.1 Router

Given a existing resources usages and a valid resource path (with unspecified entrances, exits and gates) through the infrastructure the *Router* aims to find non-conflicting usages for that path. Multiple feasible solutions may exist, but ties are broken by assigning an artificial cost to dwell times at each resource. The *Router* recursively explores all available usage windows on a single resources. If compatible windows (earliest entrance and latest exit) are found for all resources on the path then the lowest cost assignment is made (using the artificial costs). The optimal solution can be sought by continuing the search and pruning unexplored paths whenever possible.

A.5 Final Remarks

The problem has proven to be very difficult. Implementing the solution framework has been time-consuming and the work is still not completed. Routing of joint arrivals and departures has been omitted due to time constraints.